JUNE 1991 £3.25
DM. 16

Commodore ·DISK·USER·

POWER TOOLS
A Suite of 6 Utilities

THE DISK

Power Tools
I.D.O.S. V2
MONSTERS
FRANTIC
COUNTDOWN

06
9 770953 061014

LOGIC

London 081-882 4942 Cheshunt 0992 25323 Peterborough 0733 49696 London W1 071 935 2547

FULL RANGE OF AMIGA, ST, SEGA, 64, SPEC, AMSTRAD, PC, PCW, 2600, XL/XE, BBC

CAULDRON II	FIREBIRD	4.99
MUNSTERS	ALTERNATIVE	4.99
NAM	MINDSCAPE	7.99
SLOT CAR RACER	MINDSCAPE	4.99
BEYOND THE DARK CASTLE	MINDSCAPE	4.99
INDOOR SOCCER	MINDSCAPE	4.99
FINAL ASSAULT	EPYX	4.99
DIVE BOMBER	EPYX	6.99
SPY V SPY III	EPYX	4.99
SPORTS A RONI	EPYX	4.99
SPORTS NEWS BASEBALL	EPYX	5.99
DEATH SWORD	EPYX	7.99
IMPOSSIBLE MISSION	EPYX	2.99
THE GAMES SUMMER EDITION	EPYX	6.99
4*4 OFF ROAD RACING	EPYX	4.99
STREET SPORTS FOOTBA	EPYX	4.99
CALIFORNIA GAMES	EPYX	4.99
WORLD GAMES	EPYX	4.99

STREET SPORTS BASEBALL	EPYX	4.99
SPACE STATION OBLIVION	EPYX	7.99
AXE OF RAGE	EPYX	4.99
SUMMER CHALLENGE	MINDSCAPE	4.99
WINTER CHALLENGE	MINDSCAPE	4.99
TRANSFORMERS	MEDIAGENIC	4.99
CHAMPIONSHIP BASEBALL	MEDIAGENIC	4.99
LEATHER GODDESSES OF PHOBOS	MEDIAGENIC	4.99
EYE OF HORUS	LOGOTRON	3.99
STARRAY	LOGOTRON	3.99
FOOTBALL MANAGER	PRISM	4.99
ADDICTABALL	ALLIGATA	3.99
FOOTBALL MANAGER 2 CONSTRUCTION KIT	PRISM	8.95
NORTH STAR	GREMLIN	2.99

INTERNATIONAL SOCCER	CBM	4.99
VENOM STRIKES BACK	GREMLIN	2.99
GARY LINEKARS SUPERSKILLS		3.99
STARCROSS	CBM	4.99
SUSPENDED	CBM	4.99
DEADLINE	CBM	4.99
PARADROID	HEWSON	2.99
MURDER OFF MIAMI	CRL	4.99
UP PERISCOPE	SUB LOGIC	9.95
RUNNING MAN	GRANDSLAM	4.99
SUPER SCRAMBLE SIMULATOR		4.99
GLIDER PILOT	CRL	4.99
HERCULES SLAYER OF THE DAMNED	GREMLIN	2.99
BAAL	PSYGNOSIS	4.99
EXOLON	HEWSON	2.99
SHIRLY MULDOWNEYS TOP FUEL CHALLENGE	US GOLD	4.99
BLASTEROIDS	IMAGE WORKS	3.99

SOUND STUDIO HOME RECORDING STUDIO	4.95
GEOS INC GEO WRITE GRAPHICS OPERATING SYSTEM WITH WORD PROCESSOR	9.95
SOUND EXPANDER FM SOUND MODULE	9.95
ASSEMBLER DEVELOPER	9.95
PROGRAMMERS UTILITIES	9.95
LOGO	9.95
PROGRAMMERS TOOL BOX	4.95
INTRO TO BASIC PART 2	4.95
PACK OF ACES INTERNATIONAL KARATE, BOULDERDASH WHO DARES WINS, NEXUS	4.99
BEST OF ELITE BOMB JACK, FRANK BRUNOS BOXING SPACE HARRIER, AIRWOLF	4.99
POWER HOUSE FIGHT NIGHT, OSMIUM	2.99
GOLDEN OLDIES BIATHLON, MOONSWEEPER, LAWN TENNIS SPACE GALLERY, SLALOM, INTRUDER, SQUASH KO BOXING ETC ETC ETC ETC	6.99
KICK BUT SLAM FIST, RAMBO, UCHI MATA, BOP N WRESTLE	4.99
PERSONAL MONEY MANAGER	9.95

BALLISTIX	PSYGNOSIS	4.99
NINJA HAMPSTER	CRL	3.99
ARTURA	GREMLIN	4.99
APACHE STRIKE	ACTIVISION	4.99
PAZZAZ	CBM	4.99
SCRAMBLE SPIRITS	GRANDSLAM	5.99
DEJA VU	MIRRORSOFT	6.99
SOKO BAN	SPECTRUM HOL	4.99
MURDER BY THE DOZEN	CBS	7.99
PARRALAX	OCEAN	4.99
INFILTRATOR II	MINDSCAPE	5.99
THE FLINTSTONES	GRANDSLAM	4.99
MS PACMAN		4.99
THUNDER CHOPPER	SUB LOGIC	9.95
UNINVITED	MINDSCAPE	7.99
PACLAND	GRANDSLAM	7.99
TERRYS BIG ADVENTURE	GRANDSLAM	5.99
BLOODWYCH	IMAGE WORKS	6.99
FIGHTING SOCCER	ACTIVISION	4.99
NAVY SEAL	COSMI	7.99
INTRIQUE	MIRRORSOFT	6.99
BATALLION COMMANDER	SSI	6.99
THE THREE STOOGES	MIRRORSOFT	5.99

SWIFT SPREADSHEET 9.95

THE MOST POWERFULL SPREADSHEET PROGRAM
AVAILABLE ON THE COMMODORE 64, WITH POP
UP MENU CONTROL, CELL MATRIX FROM A1 TO
Z254 OR LARGER ON THE C128.

SUB BATTLE SIMULATOR 9.95

THIS IS YOUR CHANCE TO EMBARK ON WHAT IS
UNQUESTIONABLY THE MOST DETAILED, REALISTIC, ALL ENCOMPASSING WAR SIMULATION EVER CREATED.

TRACK AND FIELD 9.95

THE CLASSIC SPORTS ARCADE GAME BY ATARI
SOFT INCLUDING A FREE ARCADE STYLE CONTROL PAD.

JET 11.95

BY SUBLOGIC. FLY AN F16 FIGHTING FALCON
THE MOST ADVANCED TACTICAL FIGHTER IN
THE WORLD, OR PUT YOURSELF AT THE CONTROLS OF A CARRIER BASED F1`8 HORNET
THE US NAVYS NEWEST MULTI ROLE FIGHTER.

STEALTH MISSION 14.95

STEALTH MISSION PUTS YOU IN THE PILOTS SEAT OF THREE DIFFERENT JETS FROM THEF19, X29, OR NAVY TOMCAT,

NEW 3D ANIMATION TECHNIQUES PROVIDE DRAMATICALLYFASTER FRAME RATES FOR ALL COCKPIT WINDOW VIEWS.

INCLUDES COMPLETE COR, ILS, ADF AND DME AVIONICS FOR CROSS COUNTRY NAVIGATION.

COMPATIBLE WITH SUBLOGIC SCENARY DISCS.

WHERE TO BUY:
EITHER AT

19 THE BROARDWAY
THE BOURNE
SOUTHGATE
LONDON
N14 6PH

UNIT 6
MIDGATE
PETERBOROUGH
CAMBS
PE1 1TN

5 LYNTON
PARADE
CHESHUNT
HERTS
EN8 8LF

MAIL ORDER TO:
5 LYNTON PARADE
CHESHUNT
HERTS
EN8 8LF
Tel: 0992 25323

POSTAGE AND PACKING: 1-3 ITEMS 75P, 4 OR MORE £1.00

Commodore ·DISK·USER·

Volume 4 Number 8 JUNE 1991

ON THE DISK

IN THE MAGAZINE

Publisher: Hasnain Walji
Group Editor: Paul Eves
Designer: Mark Newton
Technical Editor: Jason Finch
Program Evaluator John Simpson
Publishing Consultant: Paul Crowder
Advertisement Manager: Cass Gilroy
Classified Sales: Deborah Curran
Distribution: Seymour Press Distribution Ltd. Winsor House, 1270 London Road, Norbury, London SW16 4DH. Tel:081 679 1899. Fax: 081 679 8907
Printed By: Gibbons Barford Print

Subscription Rates

UK	£39.00
Europe	£45.00
Middle East	£45.00
Far East	£50.20
Rest of World	£46.50

Airmail rates on request
Contact: Select Subscriptions. Tel: (0442) 876661

Commodore Disk User is a monthly magazine published on the 3rd Friday of every month. Alphavite Publications Limited, 20, Potters Lane, Kiln Farm, Milton Keynes, MK11 3HF. Telephone: (0908) 569819 FAX: (0908) 260229. For advertising ring (0908) 569819

EDITORS COMMENT

Welcome to another edition of your favourite magazine for the C64. This month we bring you a couple of games to keep you amused, and a few more serious programming tools.

The first of the Games is MONSTERS, this is a strategy type buy and sell game. The idea being to work your way up to owning the top dog. The second, FRANTIC, is a great shoot 'em up.

For the more serious user, we have an excellent suite of programs called POWER TOOLS. These utilities are all you need to create really professional programs. It includes such things as; A Linker, Autorunner, Compactor, Picture Convertor and Text Writer.

We have given you an update to an earlier CDU program called IDOS, which stands for Interactive Disk Operating System. This program will make the handling of your disk drive a lot easier.

That just about sums it all up. Hope you enjoy the issue.

DISK INSTRUCTIONS

Although we do everything possible to ensure that CDU is compatible with all C64 and C128 computers, one point we must make clear is this. The use of 'Fast Loaders', 'Cartridges' or alternative operating systems such as 'Dolphin DOS', may not guarantee that your disk will function properly. If you experience problems and you have one of the above, then we suggest you disable them and use the computer under normal, standard conditions. Getting the programs up and running should not present you with any difficulties, simply put your disk in the drive and enter the command.

LOAD"MENU",8,1

Once the disk menu has loaded you will be able to start any of the programs simply be selecting the desired one from the list. It is possible for some programs to alter the computers memory so that you will not be able to LOAD programs from the menu correctly until you reset the machine. We therefore suggest that you turn your computer off and then on again, before loading each program.

HOW TO COPY CDU FILES

You are welcome to make as many of your own copies of CDU programs as you want, as long as you do not pass them on to other people, or worse, sell them for profit. For people who want to make legitimate copies, we have provided a very simple machine code file copier. To use it, simply select the item FILE COPIER from the main menu. Instructions are presented on screen.

DISK FAILURE

If for any reason the disk with your copy of CDU will not work on your system then please carefully re-read the operating instructions in the magazine. If you still experience problems then:

1. If you are a subscriber, return it to:
   Select Subscriptions Ltd
   5, River Park Estate
   Berkhamsted
   Herts
   HP4 1HL Telephone; 0442 876661

2. If you bought it from a newsagents, then return it to:
   CDU Replacements
   STANLEY PRECISION DATA SYSTEMS LTD
   Unit F
   Cavendish Courtyard
   Sallow Road
   Weldon North Industrial Estate
   Corby
   Northants
   NN17 1JX Telephone; 0536 61787

Within eight weeks of publication date disks are replaced free.

After eight weeks a replacement disk can be supplied from STANLEY PRECISION DATA SYSTEMS LTD for a service charge of £1.00. Return the faulty disk with a cheque or postal order made out to STANLEY PRECISION DATA SYSTEMS LTD and clearly state the issue of CDU that you require. No documentation will be supplied.

Please use appropriate packaging, cardboard stiffener at least, when returning disk. Do not send back your magazine, only the disk please.

NOTE: Do not send your disks back to the above address if its a program that does not appear to work. Only if the DISK is faulty. Program faults should be sent to: BUG FINDERS, CDU, Alphavite Publications Ltd, Unit 20, Potters Lane, Kiln Farm, Milton Keynes, MK11 3HF. Thank you.

COUNTDOWN

Blank screens and time uncertainty on loading are now a thing of the past

PETER WEIGHILL

One annoyance with loading programs is the fact that you have only a rough estimation of how long it will take before loading is complete. I have always felt that a number counting down on screen, (as in all those earlier NOVALOAD programs), would reduce this annoyance as at least you can see where it is in the loading procedure. It also gives you something to watch. Having something to watch is normally achieved by loading another small program before the main one. Because of the reasons described above, I have designed "COUNTDOWN". Using the procedure outlined below, COUNTDOWN is saved together with your original program, into one complete file, adding only 5 - 6 blocks to the original length of your program. The new program will run automatically upon completion of loading.

WHAT'S IT ALL ABOUT

With COUNTDOWN installed as part of your program, you should have a screen display 5 seconds after LOADING appears on the screen no matter how big your program is. The screen display you obtain can be customised by yourself to display any information you want. E.g. Instructions about the program being loaded. The only limitation to your imagination is that it must fit into a space 6 x 40 characters and be in one colour.

Suppose you have a file called "PROGRAM". This is what you should do to use the countdown with it.

1. **LOAD"COUNTDOWN",8,1**

When COUNTDOWN has loaded, the screen will show some brief instructions, and will have "ready." at the bottom of the screen.

2. **LOAD"PROGRAM",8**

The program should start at $0801 and should also start in basic. (The maximum program length allowed is 137 blocks.)

3. If you wish to edit the 'design area' then you should type SYS49500 and press return. You should now draw your design, using lower case characters only, inside the white outline. Once complete and to your satisfaction, move the cursor to the top of the screen and press return on the SYS49610 (already printed on the screen) to store the design in memory. NOTE: The design area can only be in one colour.

4. You may load/save a design screen to disk. On the 3rd line from the top of the screen just type;

SYS51200"FILENAME",8,1 to save.

or

SYS51222"FILENAME",8,1 to load.

5. To save your new program, you should type:

SYS49152"NEW PROGRAM",a,1,b,c,d

Where: a = disk drive number
b = border/background colour
c = colour of countdown area (top)
d = colour of design area (bottom)

NOTE: c or d should not be the same as b.

6. You should then test out the program by typing LOAD"NEW PROGRAM",8,1

During loading of any program save using COUNTDOWN the following message should appear at the top of the screen.

**** TO GO UNTIL LOADING COMPLETE**

A number should appear where the asterisks are and it should start to countdown to zero while the program continues to load. At zero the program will have loaded and will automatically run. The design area should also appear on the screen about three quarters of the way down it.

HOW COUNTDOWN WORKS

The new program starts to load at $0231. During the loading, the interrupt vector is changed to $0231 (Normally $EA31). When an interrupt occurs, the code at $0231 clears the screen, alters the screen colours and changes the interrupt to $02a7. NOTE The countdown will not work properly with some turbo-load cartridges installed as they disable interrupts when loading. The program will still run normally but the countdown will not be shown while loading. Interrupt $02a7 fills the screen with colours and waits until the screen has been loaded. Then changes interrupt to $0503 (in screen memory) Interrupt $0503 begins countdown. This program uses the value stored in $AF (175) by the operating system to work out the value placed on the screen during the countdown. The operating system uses $AF and $AE to remember where to place the next byte of the program being loaded from disk.

Once loading is complete, the Operating System normally jumps to the location stored in $0326-0327 (CHROUT) to display the READY message but the value in $0326-0327 has been changed to $0334. $0334 restores O/S vectors and runs basic program. If you want to see how the countdown works you should look at $c231-c800 when COUNTDOWN is in memory. These memory positions ($c231-c800) are the same as $0231-0800 described above.

POWER TOOLS

The absolute, ultimate programming utility for C64 users
NM156

POWER TOOLS is a set of six utility programs for the Commodore 64, it is ideally suited to beginners in either MACHINE CODE or BASIC programming, but can also be used by practically anyone who has some familiarity with the operation of their computer.
The main objective when creating POWER TOOLS was to produce a disk which would contain and allow easy access to a wide range of small but powerful utilities, thus reducing precious development time and the need for endless disk searches. The six utilities in POWER TOOLS will give you a head start in this respect, but more importantly they provide you with a foundation on which you can build your own personal toolkit.

INSTALLATION

To load one of the tools:-

Insert the CDU disk, or copy all the POWER TOOL modules onto your own POWER TOOLS disk. Then enter the following commands:-

LOAD"MENU",8,1 or if you have made up your own POWER TOOLS disk, LOAD "Power Tools",8 (press RETURN) then type RUN (press RETURN)

A menu will then be displayed and you can select the required tool by pressing a key from 1-6, for example to load SCROLL WRITER simply press key 6.

To reduce loading time it is possible to load a tool without using the menu,for example you can load the LINKER straight into memory by entering:-

LOAD"01-linker ",8 (press RETURN) then type RUN (press RETURN)

INTRODUCTION

For the machine code programmer there is a PROGRAM LINKER, COMPACTER, SCRAMBLER and AUTOBOOT MAKER. Basic programmers can also make use of the AUTOBOOT MAKER, if you are an artist then the PICTURE CONVERTER could be a handy little program.

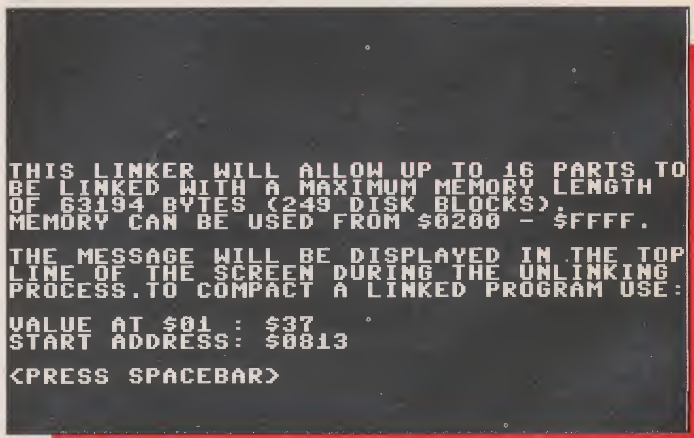
Finally, on a less serious level, anyone with or without programming ability can use SCROLL WRITER to create smooth vertical
scrolling messages.

IMPORTANT NOTES

Any numerical value which is prefixed with a dollar sign ($) means the value is in hexadecimal notation. All input/output operations are to disk (Device 8.). The author of this software does not accept any liability for any event arising from the use of these programs.

LINKER

This utility is used to join a multi-part machine code



```
THIS LINKER WILL ALLOW UP TO 16 PARTS TO
BE LINKED WITH A MAXIMUM MEMORY LENGTH
OF 63194 BYTES (249 DISK BLOCKS).
MEMORY CAN BE USED FROM $0200 - $FFFF.

THE MESSAGE WILL BE DISPLAYED IN THE TOP
LINE OF THE SCREEN DURING THE UNLINKING
PROCESS.TO COMPACT A LINKED PROGRAM USE:

VALUE AT $01 : $37
START ADDRESS: $0813

<PRESS SPACEBAR>
```

program into one file that can be loaded and run from Basic.
As a simple example of how the Linker operates, imagine you have just written a game which ends up being split into various files such as Sprite data, Character sets, Music etc. The Linker would load and store each part sequentially in memory, when all the parts have been loaded you give the Linker the start address of the game, it would then put a small machine code routine before all the parts and save everything as one file. When this file is loaded and run from basic, the machine code transfers each part to its true location then finally jumps to the start address of the game.

There are of course many alternative ways to load and run a machine code program but they each have certain drawbacks. For example, a small Basic program could be written which would load each part then a SYS command given to start the program, but amongst other things it would not look very professional and would definitely be slow at loading. Cartridge owners could load and start the program via a monitor, and then use a freezer system to stop the program on the title screen and save it as a single file but this way is prone to errors as memory can easily become corrupted, and even if the cartridge has excellent inbuilt compaction routines you will nearly always end up with a very large file, especially compared to using a Linker, so beware!

You will find then that a Linker is definitely the safest way to convert a multi-part program into a single load and run file. Note: Although the Linker was primarily designed for multi-part programs there is nothing to stop you using it to convert a single part program into a load and run file.

USING THE LINKER

When the Linker has loaded you will see four options which are all accessed by pressing one of the function keys:-

F1 - INFORMATION

This option simply displays a few helpful facts about this version of the Linker.

F3 - EDIT MESSAGE

During the process in which the parts of a linked program are transferred back to their original locations, a message of up to 26 characters can be displayed in the top line of the screen. This allows you to customise the Linker to a certain extent, by printing your own message such as the name of the Demo group you are in, or whatever. The message will only stay on the screen until all the parts have been transferred, so for small programs you can always blank the message out by filling it with spaces. To view the current message without changing it simply press F3, then RUNSTOP, to change it, type in your new message then press RETURN.

F5 - LINK FILES

This is where you will select the files to be linked, up to 16 files can be loaded giving a maximum program length of 63194 bytes or 249 disk blocks.

Each part must load into memory from 512-65535 ($0200-$FFFF), but once your program is running you are free to use all the memory. As you can imagine this makes the Linker very powerful in that it can use practically all the memory, so you should not have any problems regarding program size.

Okay, first of all make sure all the parts to be linked are on the disk in the drive, and that you have a disk available with enough space to hold the final linked program, you can approximately calculate how many disk blocks it will need by adding together each parts disk block usage and adding an extra 2 blocks for a final result.

Each program filename will be displayed on the screen, to include the file for linking press Y for yes or to see the next filename press N for no, if all the parts have been selected then you can skip the rest of the directory by pressing RETURN.

You will now be asked to enter a value (2 digit Hex) for location 1 (as most of you should know this is the input/output register which controls the memory configuration) which is important if your program happens to start under a ROM. You will now be asked to enter the actual start address (4 digit hex), then finally a filename for the linked program.

You can abandon any input by pressing the RUNSTOP key which will return you to the title screen.

The Linker will now load and sort each of the selected parts and then ask you to insert a Destination disk to save the linked program on, after which it will exit by resetting the computer via the Basic cold start routine at 64738 ($FCE2).

You can now load up the linked program by using a normal load command from Basic:-

LOAD"filename",8 (press RETURN) then type RUN (press RETURN)

Make sure that you check the linked program is fully working correctly before deleting any of the parts.

F7 - DIRECTORY

This option displays the disk directory of the current disk (device 8).

COMPACTOR

This utility can be used to reduce the size of machine code programs, thus saving disk space and loading time, however since it employs an elementary method of data compression it will be more effective on programs containing graphics data such as hires/text screens, character sets, and sprite data.

To compact a program is a simple process of loading it into the Compactor and saving the end result, which can then be loaded and run from Basic. If your program is split into parts you must first join it together as one file by using the Linker which is provided with Power Tools.

Any memory in the range 512-65535 ($0200-$FFFF) can be used by your program, and although it is impossible to give the exact size of the largest file that can be loaded (this is because compression takes place while loading) it should be able to handle most of your programs with ease.

However,if a file is to large then loading will be abandoned and you will receive the message:- Error: File To Long and then returned to the title screen.

USING THE COMPACTOR

When the Compactor has loaded you will see the title screen and two windows, one named Source and the other Destination.

First type in the filename of the program you want to compact (Source) and press RETURN,next type in the new filename you want to give the compacted program (Destination) and press RETURN.

You will now be asked to enter a value (2 digit Hex) for location 1 (which is the input/output register used to setup different memory configurations) this will be important if your program starts under a ROM. On most occasions you will probably use a value such as $37 to set the default memory map, with Basic and Kernal ROMs switched in.

Finally you will be asked for the start address (4 digit Hex) of the program, now insert the disk containing the program to be compacted and press the Spacebar, after loading and compression has finished insert a disk to save the compacted program on and press the SPACEBAR. You can now load the compacted version with a normal load command:-

LOAD"filename",8 (press RETURN) then type RUN (press RETURN)

Please check that it is fully working correctly before deleting the original program.

On certain files you may end up with a larger version even after compaction (list the directory and compare each versions disk block count to see) if so it would be wise to stick with the original version.

If you are compacting a program which has been previously linked using the linker provided with power tools then you should use the following values:-

Location $01 = $37
Start Address = $0813

Two other options are available on the title screen they are:-

F1 - INFORMATION

Will display some notes about this version of the compactor.

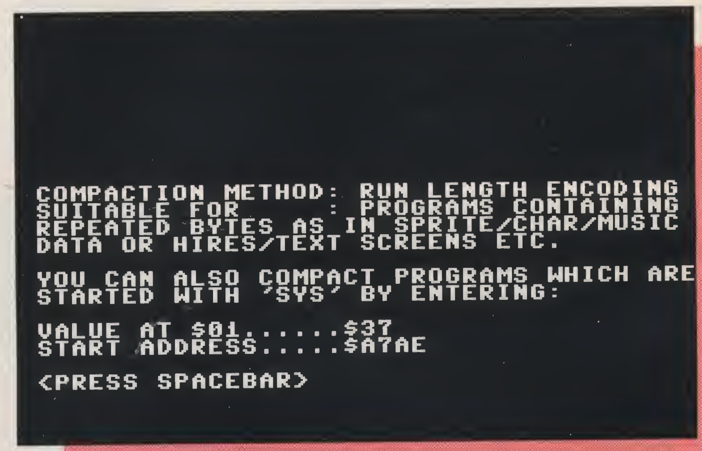
F7 - DIRECTORY

Will display the current disk directory (device 8) onto the screen.

COMPRESSION METHOD

The Compactor uses a well known method of data compression called Run Length Encoding, which works on repeated numbers.

If we consider a simple example on the numbers:- $03,$03,$03,$03,$03,$03 these could be more efficiently encoded into the following:- $A0,$06,$03 where the number $A0 is used as a marker for decompression, and the number $06 is a count of how many times the next number $03 appears. This can be more clearly defined by using an expression such as: (Marker,Count,Number), note, the marker $A0 could



```
COMPACTION METHOD: RUN LENGTH ENCODING
SUITABLE FOR     : PROGRAMS CONTAINING
REPEATED BYTES AS IN SPRITE/CHAR/MUSIC
DATA OR HIRES/TEXT SCREENS ETC.

YOU CAN ALSO COMPACT PROGRAMS WHICH ARE
STARTED WITH 'SYS' BY ENTERING:

VALUE AT $01......$37
START ADDRESS.....$A7AE

<PRESS SPACEBAR>
```

have been any other number but without it the decompression routine would not be able to distinguish between normal and compressed numbers. As you can see our original six numbers have been reduced to just three which is a saving of 50%.

AUTOBOOT MAKER

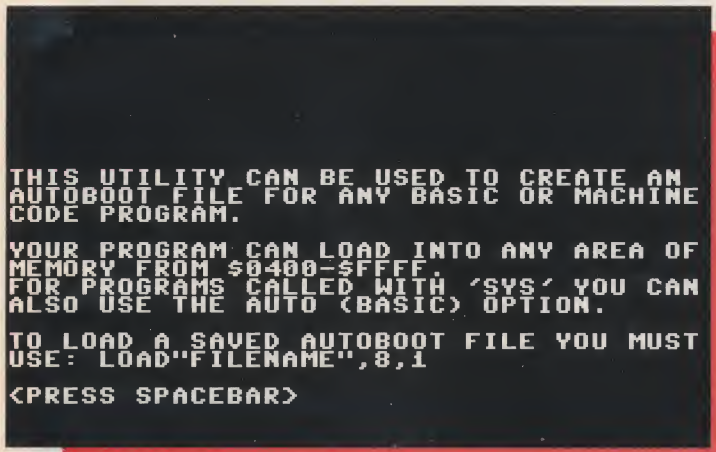
To add that professional touch to your programs, why not make them auto-load and run? This is exactly what Autoboot Maker will allow you to do for either Basic or machine code programs.

There are many uses for this type of utility, for instance, an autoboot could be the first link in a chain for loading various parts such as a hires title screen or even a fastloader.

A couple of points to remember are that the created autoboot file must be saved onto the same disk as the program you want it to auto-load and it must be reloaded by using a forced load command similar to the following:-

LOAD"autoboot",8,1 (press RETURN) then type RUN (press RETURN)

Note the extra comma and one from a normal load command, these make sure the file is loaded back into the memory from which it was saved. Okay, when the Autoboot Maker has loaded you will see four options accessed by pressing a function key..



```
THIS UTILITY CAN BE USED TO CREATE AN
AUTOBOOT FILE FOR ANY BASIC OR MACHINE
CODE PROGRAM.

YOUR PROGRAM CAN LOAD INTO ANY AREA OF
MEMORY FROM $0400-$FFFF.
FOR PROGRAMS CALLED WITH 'SYS' YOU CAN
ALSO USE THE AUTO (BASIC) OPTION.

TO LOAD A SAVED AUTOBOOT FILE YOU MUST
USE: LOAD"FILENAME",8,1

<PRESS SPACEBAR>
```

F1 - INFORMATION

This option simply displays a few facts about this version of the Autoboot Maker.

F3 - AUTO BASIC

Select this option to create an autoboot file for a Basic program, or any program which is started with an SYS command from Basic. You will first be prompted for the filename of the Basic program, make sure you enter this exactly as it appears in the directory, if the filename contains any reverse or graphic characters you will have to rename it.

You will now be prompted to enter a filename for the autoboot file,and then you can enter a message of up to 40 characters,this will be displayed in the top line of the screen while the autoboot file loads your program. If you do not want a message then just press the RETURN key, finally you will be told to insert the Destination disk (containing the Basic program) then press the Spacebar, and voila the autoboot file will be saved and you will be returned to the title screen.

F5 - AUTO M/CODE

This option must be selected if you want to create an autoboot file that will load and start a machine code program by jumping to its start address.

First you will be prompted to enter the filename of the machine code program, make sure it is entered exactly as it appears in the directory.

Next enter a value for location 1 (2 digit Hex) so that the memory will be configured correctly for the jump to the start of the program. Next enter the actual start address (4 digit Hex) and then the filename for the autoboot file,and last but not least you can enter a message of up to 40 characters that will be displayed in the top line of the screen when the autoboot file has been loaded.

Finally insert the disk containing the machine code program and press the Spacebar to save the autoboot file, that's it!

F7 - DIRECTORY

Pressing F7 will display the directory of the current disk (device 8.)

HOW THE AUTOBOOT WORKS

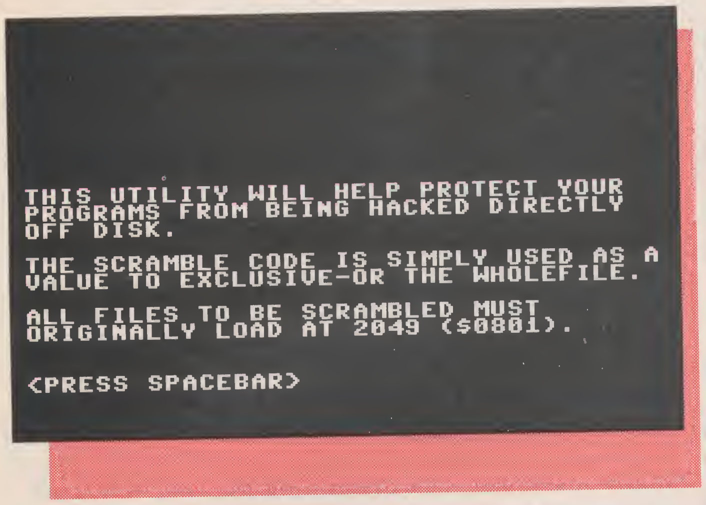
The autoboot file loads into memory at 790 ($0316) which is the Break Interrupt vector, this vector and all the system vectors from 792-818 ($0318-$0332) are set to their default address's except for the Chrout vector at 806 ($0326) which is pointed to the start of the autoboot code,this has the effect of automatically starting the autoboot without control passing back to Basic.

The first thing the autoboot code does is to reset the Chrout vector to its default address at 61898 ($F1CA), next it sets the screen & border colours and displays the message on the screen, then it loads the program using the normal kernal load routine. Finally if the program is Basic it is started with an absolute jump to the ROM routine at 42926 ($A7AE) or if the program is machine code will set location 1 and do an absolute jump to the start address.

SCRAMBLER

This utility should help protect your programs, it will not stop the experienced hacker, however it will confuse others enough to prevent them from tampering with your code.

All programs to be scrambled must originally load at 2049 ($0801) so machine code users should first put their programs through the Linker provided with POWER TOOLS, or a similar utility.



```
THIS UTILITY WILL HELP PROTECT YOUR
PROGRAMS FROM BEING HACKED DIRECTLY
OFF DISK.

THE SCRAMBLE CODE IS SIMPLY USED AS A
VALUE TO EXCLUSIVE-OR THE WHOLEFILE.

ALL FILES TO BE SCRAMBLED MUST
ORIGINALLY LOAD AT 2049 ($0801).

<PRESS SPACEBAR>
```

The Scrambler can load programs up to a maximum length of approx 237 disk blocks which should be adequate for most of your programs,however if you exceed this then the loading will be abandoned and the message - (Error:File Too Long) will be displayed and you will be returned to the title screen.

USING THE SCRAMBLER

When the Scrambler has loaded you will see the title screen and two windows, one named Source and the other Destination.

First of all type in the filename of the program to be scrambled (Source) and press RETURN, next type in the new filename you want to give the program after it has been scrambled (Destination). Okay now enter any value (as a 2 digit Hex number) between $01-$FF to be used as a scramble code.

Finally, insert the disk containing the program to be scrambled and press the Spacebar, after it has been loaded, insert a disk to save the new scrambled program onto and again press the Spacebar to save and complete the process. You should now load up the scrambled version and check it is fully working just like the original.

There are two other options available on the title screen which are:-

F1 - INFORMATION

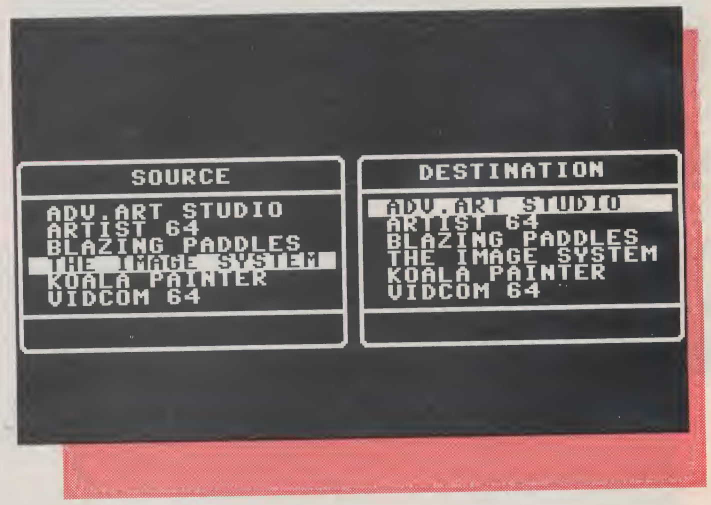
Simply displays a few helpful facts about this version of the Scrambler.

F7 - DIRECTORY

Will display the disk directory of the current disk (device 8.)

SCRAMBLING METHOD

The method used to scramble the file is very simple, if you are a machine code programmer then you have no doubt come across the instruction EOR (Exclusive-Or) which can be used to invert any bits in a byte. So if we were to exclusive-or a byte twice using the same value then we would end up with the original byte. Hence the Scrambler just uses the scramble code to exclusive-or the whole file which is then saved, when the program is Run a small machine code routine exclusive-or's the file again using the same scramble code thus returning it to its original state. Note, you can achieve added protection by scrambling a program more than once, but each scramble will increase the programs size.



```
        SOURCE                  DESTINATION
 ADV.ART STUDIO           ADV.ART STUDIO
 ARTIST 64                ARTIST 64
 BLAZING PADDLES          BLAZING PADDLES
 THE IMAGE SYSTEM         THE IMAGE SYSTEM
 KOALA PAINTER            KOALA PAINTER
 VIDCOM 64                VIDCOM 64
```

PICTURE CONVERTER

Picture Converter is a small but powerful utility which will convert multicolour hires screens between six of the best available art packages on the Commodore 64. If for example you have in the past been using Koala Painter and then recently purchased the Advanced Art Studio you will have no doubt discovered it is impossible to load a picture done with Koala into the Art Studio and vice versa. If you have come across this annoying problem then fear no more because Picture Converter will allow you to do this and is very easy to use. Conversions can be carried out between the following art packages:-

ADVANCED ART STUDIO
ARTIST 64
BLAZING PADDLES
THE IMAGE SYSTEM
KOALA PAINTER
VIDCOM 64

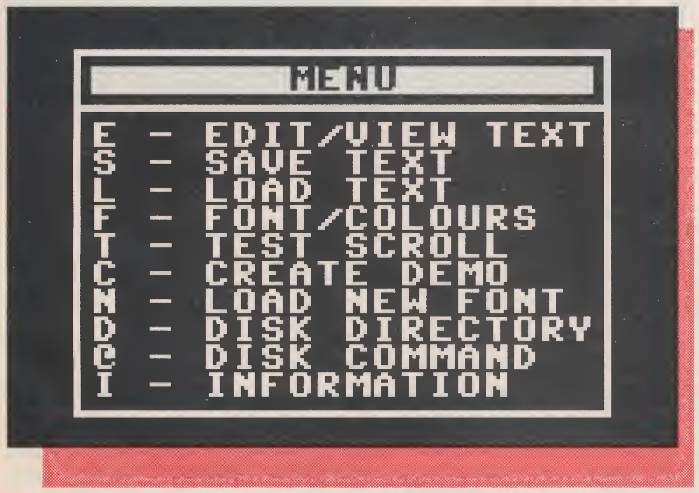
USING PICTURE CONVERTER

When the Converter has loaded you will see two windows titled Source and Destination, inside both windows are the names of the aforementioned art packages. Use the F5 & F7 keys to highlight the conversion you require, then press RETURN. For example if you wanted to convert Koala picture into Blazing Paddles format, you would simply locate Koala Painter in the Source window and Blazing Paddles in the Destination window.

You will now be prompted to enter the filename of the Source picture to load, and then the converted Destination picture, it is important that these are entered correctly, and to help in this there is an option to display the disk directory by pressing F1.

As you will probably know, each art package has its own special set of characters included with the pictures filename (e.g. Blazing Paddles uses the prefix PI.) one of the reasons for this is to help distinguish pictures from other files, in any case you can rest assured that these are taken care of and you need only enter the filename exactly as you did when saving the picture from the relevant art package. Note, there is one exception to this, for Koala Painter pictures you must also enter the letter which appears after the word PIC, then a space and then the filename, if you are not sure about this then listing the directory by pressing F1 should help.

After entering the filenames just follow the on screen instructions and the picture will be loaded, converted, and saved in the new format.



```
MENU
E - EDIT/VIEW TEXT
S - SAVE TEXT
L - LOAD TEXT
F - FONT/COLOURS
T - TEST SCROLL
C - CREATE DEMO
N - LOAD NEW FONT
D - DISK DIRECTORY
@ - DISK COMMAND
I - INFORMATION
```

SCROLL WRITER

Scroll Writer will allow you to create a stand alone demo which includes a smooth vertical scrolling message and optional background music. It can be used for a variety of applications, at the simplest level you could communicate via the computer to other Commodore users, or if you are fortunate to own a shop, you might use Scroll Writer to advertise your latest products and prices. Even though Scroll Writer is very easy to use, to get the best results requires you to have a basic knowledge of Commodore 64 character graphics, so it may be useful to purchase a good reference book, you could even try your local library and save the expense.

USING SCROLL WRITER

When Scroll Writer has loaded you will see the main menu containing 10 options, to select an option simply press the corresponding key to the left of the option name.

E - EDIT/VIEW TEXT

Selecting this option takes you to the text editor, which is used to type the scrolling message. You will see a window, the right side of the window is the editing area (you should see a flashing cursor) while the left displays most of the editing keys and their function. The cursor keys are used to move around the editing window and you can scroll text into view by moving the cursor to the top or bottom of the window. To help locate your position in the text the current cursor line number (L) and column (C) are displayed at the top of the editing window. You may be wondering why you are only allowed 20 character per line, well, if you realise that there can be up to 40 characters on a line of a normal screen, and that the message will be scrolled using a font which is 2 characters wide, we calculate 40/2 = 20. There is a maximum of 200 lines for text this means you have 4000 characters which should be adequate for most applications.

The editing keys provide the following functions:-

HOME	Moves the cursor to line 1, the first line of text memory.
CLR	Moves the cursor to line 200, the last line of text memory.
F7	Displays the next page of text, unless you are on he last page.
F5	Displays the previous page of text, unless you are on the first page.
RETURN	Positions the cursor at the start of the next line.
INST	Inserts a space at the current cursor position.
DEL	Deletes the character to the left of the cursor.
F3	Toggle wordwrap feature.
F8	Aligns the current line to the left, so that text starts in column 1.
F6	Centres the text on the current line.
F2	Erases all the text.

You can also change the text colour, just like in Basic (e.g. pressing CTRL and 2 sets the text colour to white). It is also possible to centre or left align all the text in one go, to do this simply move the cursor to line 200 (press CLR) then press either F6 or F8 respectively. One feature not yet mentioned is wordwrap, although anyone who uses a good word processor regularly should need no introduction to wordwrap, a short explanation follows for everyone else. The idea behind wordwrap is to stop

words being split over lines, the editor does this automatically for you by quickly copying a split word onto the start of the next line and advancing the cursor to its correct position, this enables you to carry on typing without the need to watch for the end of the line. There is of course a bit more to it than that, so if you have not quite grasped the operation then just try typing a long word near the end of a line and watch what happens. You can switch the wordwrap feature on or off by pressing F3,when it is on you will see (ww) displayed between the current line and column numbers. Be very careful when editing your text using the left align, centre, insert and delete functions because wordwrap only operates while your typing, so it is very easy to mess up the format of your text. One key not yet explained is the left arrow key (at top left of keyboard) which is a very important character in the text as it signifies the end of the scrolling message (all further characters will be ignored) and causes the scroll to wraparound. So if you do not want the message to be repeated do not include any left arrow characters in the text.
To return to the main menu just press RUN STOP, any text will remain in memory for further editing.

S - SAVE TEXT

This option will save all the text memory, just enter a filename and press RETURN. It can be very useful if you have some unfinished text that you wish to continue at a later date.

L - LOAD TEXT

Use this option to load any text which has been previously saved using the 'Save Text' option, just enter the filename and press RETURN. Caution must be taken as any text currently in memory will be erased.

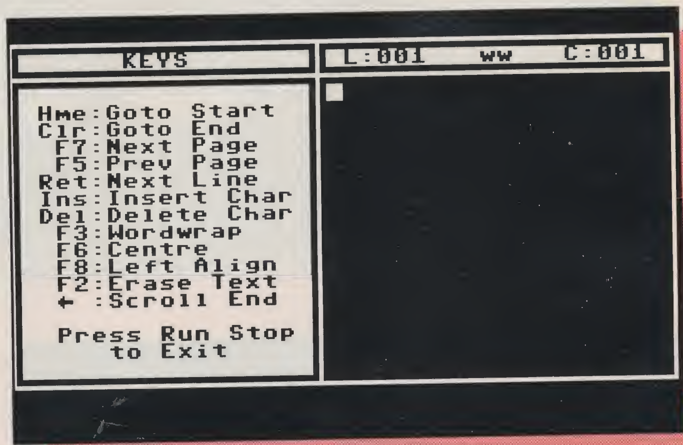
F - FONT/COLOURS

This option displays the 2x2 font that will be used to scroll the message, you can also change the colours, select from 1 of 4 inbuilt fonts and set various parameters. In the top half of the screen you will see the font, while the bottom half shows the options and current settings. All the colours and options can be changed by pressing the corresponding key on the left, to exit back to the main menu just press RUN STOP.
The current colours are displayed along with their corresponding colour code, most of these should be self explanatory, the other options however will need a few more details.
The option 'B - Blank Lines' allows you to force a blank line between each line of the text during the scroll, this was included because some fonts can look rather untidy and unreadable when printed close together.
The option 'S - Scroll Speed' sets the speed of the scroll, where 1 is the fastest, a setting of 2 was thought to be the most reasonable for reading the message, although the other speeds are there if you need them. If you press key 'C' to change the font then the names of 4 fonts will appear in the options window. When you select one of these the font will appear in the top half of the screen, as it is impossible to describe here what each font looks like, just select each one to see them all.

Note, when you make a selection, multicolour will be automatically set to the correct mode for that font, however for font B make sure that the NORMAL COLOUR is between 8-15 as it is a multicolour font.

T - TEST SCROLL

Once you have entered a message, selected a font and defined the settings you can use this option to see what the final scroll will look like.
To return to the main menu at anytime, just press the SPACEBAR.



```
        KEYS              L:001    ww    C:001

Hme:Goto  Start
Clr:Goto  End
 F7:Next  Page
 F5:Prev  Page
Ret:Next  Line
Ins:Insert  Char
Del:Delete  Char
 F3:Wordwrap
 F6:Centre
 F8:Left  Align
 F2:Erase  Text
 ← :Scroll  End

 Press  Run  Stop
    to  Exit
```

C - CREATE DEMO

This is the main option as it allows you to save out your scrolling message (as seen using Test Scroll) as a separate program which can be loaded and run from Basic.
You will first be asked if you want to include any music with your demo, if you cannot program in machine code and have no knowledge of music then you should press the 'N' key for no (or alternatively read the DISK EXTRAS section of this manual) in which case you will be asked for a filename and your demo will be saved to disk. It will take up 33 disk blocks and can be loaded using the command:-

LOAD"filename",8 (press RETURN) then type RUN (press RETURN)

Pressing the SPACEBAR during the demo will take you back to Basic.

If you want to include a piece of music with your demo then the following restrictions must be adhered to:-

The music must be in one file and playable under an IRQ Interrupt routine being called every frame (1/50th second). Also important is the fact that the music will not be played from within Scroll Writer but will be loaded and saved with your demo, this allows a greater range of memory in which your music can exist. The only memory that should not be used is from 1024-10239
($0400-$27FF) but all of zero page and memory from 10240-65535 ($2800-$FFFF) is free for your music.
Okay, once the music has been loaded you will be asked to enter the start address (4 digit Hex) of a routine to INITIALISE the music (this could do things like set the volume or tune number etc..) If you do not need to initialise the music then you must use the alternative address $0907 (or any RTS instruction.) Next you will be asked for the MUSIC PLAY address (4 digit Hex) that will be called under interrupt to play the music. Finally, enter a filename and the complete demo will be saved.

Make sure you save any text if you are creating a demo with music, just as a precaution in case you make a mistake and the demo does not work.

N - LOAD NEW FONT

This option lets you load and use your own fonts in Scroll Writer, all fonts are 2x2 characters in size and must be designed in the following format:-

Character A - Screen Codes

001 065
129 193

Where 001 is the top left of character A in the font.

D - DISK DIRECTORY

Select this option to display the disk directory of the current disk (device 8.)

@ - DISK COMMAND

Lets you send a disk command or just read the disk status by pressing RETURN.
Example of a valid command:-

S:DEMO

Scratches the file called DEMO. Consult your disk drive manual for further commands.

I - INFORMATION

This option will display a few pages of helpful facts about this version of Scroll Writer.

DISK EXTRAS

On the CDU disk you will find two extra programs called:-

SAMPLE FONT
SAMPLE MUSIC

Both of these are for use with Scroll Writer, the sample font file, if you have not already guessed, is an extra font that can be loaded into Scroll Writer using the LOAD NEW FONT option.

This font is an exact copy of the upper case character set in the Commodore 64,except it has been enlarged to a 2x2 size. If you do load this font then remember to select the FONT/COLOURS option to see the font, set multicolour to NO, and select some suitable colours.

If you have a character editor or similar utility then you could use the sample font as a base for designing your own fonts.

The 'Sample Music' file contains 3 tunes that can be used when you create a demo from Scroll Writer. Before you use the music, you might like to listen to it first, which you can do from Basic by entering the following:-

LOAD"sample music",8,1 (press RETURN) then type SYS 52800 (press RETURN) now select a tune by entering:-

POKE 52882,TUNE NO (1-3)

You can change to any other tune while the current one is playing simply by POKEing 52882 with the new tune number. If you listen to tune 1 you should find it is the title tune from POWER TOOLS.

To include the music in a demo you must use the following address's:-

INITIALISE MUSIC

Tune 1 = $CE94
Tune 2 = $CE9A
Tune 3 = $CEA0

All 3 tunes use the same music play routine:-

MUSIC PLAY

Tune 1,2,3 = $CEB0

That just about wraps everything up, I would just like to say that I hope you enjoy using POWER TOOLS as much as I had in developing it. Look out for more NM156 programmes coming your way.

BASICS OF BASIC

BASICS OF BASIC

The tutorials for the beginner in Basic continue

By John Simpson

Last month we touched very briefly on the keyword GET. This will return a character value of the last key pressed by a user. Let us examine this command in somewhat more detail and discover exactly how we can use it to our best advantage. Remember, last month, I told you how GET returns a single character ASCII string, and that a NULL string, < "" > represents no key having been depressed. We can use the null string to create a loop which will, in effect, keep testing for a key input, then, once we have a key input we can act directly upon this.

USING GET

The following line sets up our key check and continues looping until a key has been struck:

```
10 GET A$:IF A$ = "" THEN 10
```

we could have used any string variable we desired here, I chose, purely arbitrarily, A$. The IF...THEN decision tests A$ and until it contains something other than a null string it will continue to execute line 10. Whilst A$ = "" is considered to be true, but as soon A$ holds a character value other than null then the truth of the IF...THEN statement becomes false and so execution will NOT carry out the ...THEN part of the decision argument, but pass on to the next instruction which will be on the following line.

Once A$ does hold a string value we are at liberty to do several things with this information. We can test it against other characters and act upon the result, or we can change the character value into a numeric value which corresponds to its ASCII value (see page 146 of the Commodore User Manual). Let us examine a few examples:

E.G.1

```
10 GET A$:IF A$ = "" THEN 10
20 IF A$ = "Y" THEN 50
30 IF A$ <> "N" THEN 10
40 PRINT "RESULT OF GET = 'NO'.":END
50 PRINT "RESULT OF GET = 'YES'.":END
```

E.G.2

```
10 GET A$:IF A$ = "" THEN 10
20 IF A$ = "1" THEN 100
30 IF A$ = "2" THEN 200
40 IF A$ = "3" THEN 300
50 GOTO 10
100 PRINT "MENU ITEM #1"
110 END
200 PRINT "MENU ITEM #2"
210 END
300 PRINT "MENU ITEM #3"
310 END
```

The YES/NO situation often occurs within programs and a simple way of obtaining the user response is demonstrated in E.G.1. Line 20 tests A$ to see if it equals the 'Y' key and if it does branches to a part of the program where this response is dealt with. In the above example it simply directs program control to line 50 and prints the result of the key input. Line 30 deals next with any other key on the keyboard except 'N'and 'Y' (we know that 'Y' has not been pressed otherwise we would not have reached line 30). So here we test A$ to find out if it does NOT equal 'N'. If the result is true, i.e. it does not equal 'N' the program is simply diverted back to line 10 to await another key press. If the result is false and A$ does equal 'N' then, the same as line 10, the ...THEN part of the statement is not executed and control drops through to 40 which prints the response to the key press. You will note that both line 40 and 50 terminate the program.

The program example E.G.2 is a typical situation where a menu item is being looked for. In this program it assumes there is a menu with three items in it. To go to each menu option requires the user to input the menu number, either 1, 2, or 3. The important point of this program is line 50 which directs control back to line 10. To reach line 50 means that each IF...THEN decision statement was false and the numerals 1, 2, and 3 have not been struck.

By using logical methods such as these two examples you are able to restrict the user response to exactly the key or keys you desire, ignoring any others.

Now let us look at a further example where by requesting the user to strike any numerical key to activate an option or a menu item we can tighten up our code and make it more efficient.

```
10 GET A$: A = VAL(A$)
20 IF A <1 OR A> 4 THEN 10
30 ON A GOTO 100,200,300,400
100 PRINT A: GOTO 10
200 PRINT A: GOTO 10
300 PRINT A: GOTO 10
400 PRINT A: GOTO 10
```

In this simple example I have introduced another two keywords, together with a new programming technique. Let us examine this more closely.

Line 10 - the first statement on the line is the standard GET statement, but the second statement deviates from testing for a null string as we did before. Here we have assigned a number variable, A to equal the value of A$. Now the key word VAL will return a numeric VALue of A$ so long as A$ is a numerical key in the range 1 to 9 every other key will return a zero.

Line 20 - here we have set up an IF...THEN decision statement. If the value held in A is less that 1 or is greater than 6 then the wrong key has been pressed so the program will go back to the GET statement of line 10.

LINE 30 - this line introduces a new keyword and a different programming method. Instead of having several lines of separate IF...THEN statements (in this program four, 1 - 4) we can reduce this to just one line using the ON command. The ON keyword takes the value of a numerical variable from 1 upwards (you cannot use ON 0). In the above program segment I chose to place the VALue of A$ into the variable A. So, if the user presses the '1' key then the program will jump to line 100, '2' to line 200, and so on where it simply prints the value of A, the key you pressed, it then goes back to line ten for another test. You can easily see the advantage of using the ON command when there are a number of IF...THEN statements to be executed as we might find in a menu which uses the numerical keys for option selection.

We will be looking further into the On statement later when we start creating more substantial programs.

Finally, for now, we can also use the GET keyword to input multiple character strings where the use of the return key determines the string end.

```
10 GET A$: IF A$ = "" THEN 10
20 IF A$ = CHR$(13) THEN 60
30 N$ = N$ + A$
40 PRINT A$;
50 GOTO 10
60 PRINT CHR$(13)N$
```

Line 20 - this line tests to find out if the return key has been pressed and if it has then execution unconditionally jumps to line 60. CHR$(13) = RETURN.

Line 30 - just the same as the addition of numbers, we can also add together strings. Here N$ starts off as a null string and each time a character is returned to A$ during the get statement it is added to N$.

Line 40 - this simply echoes to the screen the last input held in A$, so that we can actually see our keypress. Note the use of the semicolon to stop the newline after each iteration of the print statement.

Line 50 - reverts execution back to line 10 for the next return of A$ from the user.

Line 60 - will first perform a carriage return then print the content of N$. We had to use a RETURN here to break the semicolon instruction of line 40 which held the multiple characters from A$ together on one line.

GENERATING RANDOM NUMBERS

The keyword for the random function is RND(A), where A is a the argument of RND. This will return a floating point random number from 0.0 to 1.0.

Try this short program to test the effects of RND.

```
10 PRINT RND(1)
20 X = X + 1 : IF X < 10 THEN 10
RUN

 .320315719
 .593751252
 .550784886
 .0234395862
 .363281548
 .894533694
 .105469763
 .632815719
 .761720359
 .320312619

READY.
```

This is useful if you want to simulate a tossed coin, for example. Any number equal to or below 0.5 represents a 'head' and any number above 0.5 could represent a 'tail', although this will be bias towards a head! However, if you required to simulate the fall of dice, then things are somewhat trickier. We need a method in which we can obtain whole integers, and not only that but integers which are either up to certain values, or between two values.

If you try a larger value in the RND (<argument>) not much seems to change. If you choose an argument which is a minus number, i.e. -1 then the randomness seems to be fixed. However, if you multiply RND(1) with a number you will then obtain numbers greater than 0.9999999.

```
10 PRINT 6 * RND(1)
20 X = X + 1 : IF X < 10 THEN 10
RUN

3.0726354
 2.78308138
 5.20487632
  .0782542169
 3.67820895
 4.6730914
 1.98361003
  .22367779
 5.933305737
 1.88376328

READY.
```

Looking at the number before the decimal point, they are:

3, 2, 5, 0, 3, 4, 1, 0, 5, 1

The members of the set are (0,1,2,3,4,5)

But if we are simulating dice then we need (1,2,3,4,5,6). Okay, simply add 1 to the numbers and we have simulated a dice throw. If we wanted a use a pack of cards we would need to multiply by 52. So, our RND(1) function can now be stated thus:

6*RND(1)+1 or 52*RND(1)+1 or 100*RND(1)+1.

The next stage is to get rid of the decimal point and the numbers which follow it until we arrive at an integer number.

THE INT FUNCTION.

The effect in INT(X) is to return the whole (or integer) part of the number X. The effect of the INT keyword is to leave off the fractional part. If the expression is negative any fractional part causes the next lowest negative integer to be returned.

INT(5,63981) = 5
INT(3.2) = 2
INT(1) = 1
INT(-2.467) = -3

With the INT(n) keyword we can generate whole numbers. Let us now look at the required random function:

INT(6*RND(1)+1) or INT(52*RND(1)+1)

However, just to confuse you a little bit more, the C64 basic will not accept this syntax, we need to keep the same form of 'template' but change the syntax slightly, namely:

INT(RND(1)*6)+1 or INT(RND(1)*52)+1

Great, now we can generate random numbers between 1 and n. How do we obtain a random number between, say 0 and 50, or between 100 and 200?

Here are some examples of the RND function together with their correct syntax:

```
10 X=RND(1) : REM THIS RETURNS A
FLOATING POINT NUMBER BETWEEN 0.0
AND 1.0

10 PRINT INT(RND(0)*100) :REM THIS
RETURNS RANDOM INTEGERS FROM 0 TO
99

10 X=INT(RND(1)*6)+INT(RND(1)*6)+2 :REM
SIMULATES THROWING TWO DICE

10 X=INT(RND(1)*1000)+1 : REM RETURNS
RANDOM INTEGERS FROM 1 TO 1000

10 X=INT(RND(1)*150)+100 : REM RETURNS
RANDOM INTEGERS FROM 100 TO 249

10 X=INT(RND(1)*(U-L))+L : REM RETURNS
RANDOM INTEGER NUMBERS BETWEEN
UPPER (U) AND LOWER (L) LIMITS
```

Why not write a simple print program and use the above random examples to get the feel of them.

STRINGS A SLIGHT RETURN

Now is the time to get back to 'strings' and to develop our knowledge of them together with enhanced methods of 'string' manipulation.

CUTTING STRINGS

We discovered earlier in the series that a 'string' is a group of characters enclosed within quotation marks. For example - "this is a string", and it consists of 16 characters. It is possible for us to cut up the string into multiple parts, and isolating separate characters or groups of characters from within the string. There are three basic commands for doing thus and they are:

LEFT$, RIGHT$, and MID$

Let us look a little more closely at LEFT$.

```
10 X$ = "CUTTING"
20 PRINT LEFT$(X$,3)
RUN
CUT

READY.
```

Here you can see that we have cut out the first three characters from the left side of X$, namely CUT(ting). Now we can take out the right side characters, THUS:

```
10 X$ = "CUTTING"
20 PRINT RIGHT$(X$,4)
RUN
TING

READY.
```

A program can tell a story better than a thousand words!

```
10 REM *** CUTTING UP STRINGS ***
20 X$ = "ABCDEFGH"
30 PRINTTAB(10)"1234567890"
40 FOR C = 1 TO 8
50 PRINT C TAB(10) LEFT$(X$,C)
60 NEXT C
RUN

          1234567890
1         A
2         AB
3         ABC
4         ABCD
5         ABCDE
6         ABCDEF
7         ABCDEFG
8         ABCDEFGH

READY.
```

In this program we have set up X$, printed a scale (line 30), and set up a FOR...NEXT loop of eight iterations. Within the loop at line 50 we have instructed the computer to print a vertical scale using the loop counter (C), and to perform a tabulated print under our horizontal scale of the leftside of X$ offset by the loop counter C. Just glancing at the printed output tells the story!

If we now change line 50 to this:

```
50 PRINT C TAB(10) RIGHT$(X$,C)
```

then the print out on RUN is:

```
          1234567890
1         H
2         GJ
3         FGH
4         EFGH
5         DEFGH
6         CDEFGH
7         BCDEFGH
8         ABCDEFGH
```

We can readily see that RIGHT$ has 'chopped up' up the string in the reverse order to LEFT$.

In these examples we have used LEFT$ and RIGHT$ to cut sections off either end of a string. However, we may require a section from the middle of a string, for example MM in a string of DDMMYY (day,month,year). MID$ solves this problem for us.

```
MID$(P,L)
    | |_______Length of sub string from Position (P).
    |_________Position within the string to extract from.
```

```
10 X$ = "COMMODORE"
20 PRINT MID$(X$,5,3)
30 PRINT MID$(X$,6,4)
RUN
ODO
DORE

READY.
```

```
10 REM *** CUTTING UP STRINGS ***
20 X$ = "ABCDEFGH"
30 PRINTTAB(10)"1234567890"
40 FOR C = 1 TO 8
50 PRINT C TAB(10) MID$(X$,C,2)
60 NEXT C
RUN
          1234567890
1         AB
2         BC
3         CD
4         DE
5         EF
6         FG
7         GH
8         H

READY.
```

You will see that the final iteration of the loop has only printed "H". What has occurred is that the computer has terminated with a 'null' string in the final position.

HOW LONG IS THE STRING?

In the above example programs we have known the length of the string on each occasion, however, in the real world of computing this is not always the case. We require a method to determine how long a string is, and happily just such a keyword exists. Good old, LEN.

```
10 INPUT "ENTER A CHARACTER STRING";A$
20PRINT"[CD]THESTRING,"A$",
IS"LEN(A$)"CHARACTERS LONG"
RUN
ENTER A CHARACTER STRING? JOHN SIMPSON

THE STRING JOHN SIMPSON IS 12 CHARACTERS
LONG

READY.
```

Notice that when the LENgth of the string was computed it included the space between JOHN and SIMPSON.

ANOTHER VALUE

We can also find the ASCII value of the first character of a string by using the keyword ASC(<string>). Here are some examples.

```
10 A$ = "A"
20 PRINT ASC(A$)
RUN
65

READY.

10 A$ = "ZOLTAN"
20 PRINT ASC(A$)
RUN
90

READY.
```

or alternatively to find the ASCII value of a character:

```
PRINT ASC("Z")  <CR>
90

READY.
```

which, if you do not have a copy of the ASCII character set, will allow you to create one of your own!

CALLING ALL SUBROUTINES

A subroutine is a section of often used code. For example you might want to update a display every few seconds and so you would place the code to do the update somewhere within your program and the last keyword, or command, of that particular block of code would be, RETURN. But you need to 'call' that block of code, and this is done by using the statement, GOSUB followed by the particular line number the block starts from. What this means, basically, is, GO TO A SUBROUTINE AND SAVE THE RETURN ADDRESS.

Once execution of the block of code, or more commonly, the subroutine has reached the instruction RETURN then the program will revert back to the instruction which follows the original GOSUB.

Here is a graphic example of just what takes place, lines 40 and 50 represent further lines of code:

```
          10 INPUT A
          20 INPUT B
          30 GOSUB 100 >-------
    --->-40  .....        |
    |     50  .....       |
    |                     |
    |                     |
    |   ----------<--------
    | |
    | |-> 100 PRINT "A","B"
    |      110 PRINT A,B
    |      120 RETURN ->-----
    |                     |
     ----------<-----------
```

TIME FOR TIME

In this last section for this month lessons, we shall explore the 64s built in clock, and finish with the construction of a small program which will display a real-time digital clock. In constructing the digital clock we will use input commands, string commands, formatting commands and the use of subroutines. But first let's take a look at accessing the clock itself.

Within the murky depths of the C64 lies a 'jiffy' clock, or, more technically a hardware interval timer (actually a quartz timer is used). When you power-up the system this timer is set to zero (initialised). Each tick of the timer takes place every 1/50th of a second. If you type this line in direct mode:

```
PRINT INT(TI/50)"SECONDS SINCE POWER UP"
<CR>
```

You will be informed of how many seconds have passed since you switched on the machine. A little thought and you can change the seconds into hours, minutes, and seconds.

The keyword here is TI for TIme.

STRINGING OUT TIME

The TI$ timer looks and works just the same as a real clock so long as the computer is powered-on. The 'jiffy' clock is exploited to update the value of TI$. TI$ is a string of six numeric characters in the form HHMMSS hours, minutes, and seconds. The useful thing about TI$

is that we can assign the string with a starting point similar to the way in which we might set a digital time piece. It does have a drawback (what doesn't?), and that is that it loses accuracy after tape input/output (still, we are all disk users aren't we?)

Let us look at an example of the TI$ function:

```
10 TI$ = "000000"
20 PRINT TI$
30 GOTO 20
```

If you type in and run this example you will see that the right-most digits are updated every second. If you watch for long enough you will see that when the two right-most digits reach 60, the following second they will revert to 00 and the mid-two digits will change to 01. Keep on watching and when 60 minutes go by the two left-most digits will show the hours updated. If you watch for twenty-four hours then everything will reset back to 000000. After the twenty-four hours vigil you can hit the RUN/STOP key and stop the program!

We could, of course change line ten to this, TI$ = "235955", and watch for only five or six seconds.

Okay then, to finish here is our project: Create a digital time display.

There are several aspects to think about when we commence with developing a program from scratch and it is best to outline what it is that we desire the program to achieve. Basically this will become a problem crunching exercise. By that I mean we start off with a problem and slowly break the problem down into smaller sub-problems which will eventually become the commands of the program.

So, first let us define the problem: Display a real-time digital clock where the user can set the time. Make the clock a 12 hour clock rather than 24, and allow the user to set an alarm.

From this we can now commence to create an 'algorithm' - the basis of the program.

```
1. set up screen/background colours, and
initialise any variables
2. initialise clock time
3. request if alarm required
4. if no goto 6
5. initialise alarm and time
6. print necessary screen graphics for clock
7. execute main computing loop
8. execute alarm if on and time
```

From our first outline we can see that the program will break down into several separate sections, or routines. For example most of the routines will only be executed once. This will be when the user actually sets the time and sets the alarm.

So let us look in more detail at 2 to 5. We can see that if the user wishes to set an alarm, then we will require two sets of time input. A) to set the time, and B) to set the alarm. Therefore one set of input routines will suffice for both operations, but we must be able to separate them to serve two separate tasks. Also, we can note that the user has the option to either set the alarm or not. Taking this point further then, means we need a variable to use as an indicator whether the alarm is on or off. From this we can expand our algorithm.

```
1. set up screen/background colours, and
initialise ALarm = 0 (off)
2. input if am or pm? AM$
3. input hour? H$
4. input minute? M$
5. input second? S$
6. store results in TI$
7. set alarm?   yes/no
8. if yes then ALarm = 1 : do 3,4,5, store
results in AL$
9. print necessary screen graphics for clock
10. execute main computing loop
11. execute alarm if on and time
```

Looking at number 9 we can see that we do not need to enlarge upon this. When we tackle this sub-problem it will be a matter of deciding what graphic we require.

However 10 is the main computing loop and we will need to break this down further. First we will need to cut the TI$ into three sections, HHMMSS, and once done we will need to test HH to determine when it reaches 13 the moment it does we must test AM$ and reverse it, in other words if it is PM change to AM and if AM change to PM. Next we must test AL if is not zero then we know the alarm is switched on so we must make a comparison between TI$ (the real time) and AL$ (the alarm time) If they are equal then activate an alarm system, if not ignore it. Of course once the alarm is activated we will need a method for the user to deactivate it. So, once again we can expand our algorithm.

SET UP AND INITIALISE

1. set up screen/background colours, and initialise ALarm = 0 (off)
2. input if am or pm? AM$
3. input hour? H$
4. input minute? M$
5. input second? S$
6. store results in TI$

SET ALARM

7. set alarm? yes/no
8. if yes then ALarm = 1 : go to subroutine (3,4,5) on return store results in AL$

PRINT INITIAL CLOCK DISPLAY

9. print necessary screen graphics for clock

MAIN COMPUTING LOOP

10. if LEFT$(TI$,2) = "13" then go to am/pm change, subroutine (17)
11. H1$ = LEFT$(TI$,2):M1$ = MID$(TI$,3,2): S1$ = RIGHT$(TI$,2)
12. print the strings to the screen graphic clock
13. if AL=1 then compare TI$ and AL$ if yes AL = 2 so go to subroutine (16)
14. if AL=2 then switch off alarm yes/no; if yes AL = 0

15. continue executing main computing loop - jump to (10)
EXECUTE ALARM
16. execute alarm - return to (14)
SWITCH AM/PM
17. do am/pm change; return to (11)

Now you can see how the program is slowly forming and the large problem - create and display a digital clock - is being broken down into more manageable bite sized chunks.

The next stage is to start converting our algorithm into program code. Now it just so happens that I prepared a program earlier!

On this months disk is a program BB.EG#1. If you load it into your machine, then we can run through the listing together - however, if you feel you would like to develop your own program, please do so.

THE DIGITAL CLOCK PROGRAM

Line 10 Our pokes to screen and border colour.

Line 20 this redirects program control to line 200, but saves its return address, which will be line 30, Let's go to line 200

Line 200 here we get the user to input whether it is morning or afternoon, and store the result into AM$. Note the decisional IF...THEN which ensures the user does not input more than two characters. Can you add to this and ensure that the user must input two characters?

line 210 I have concentated the RVS ON embedded character with the users input, so that later during print out the AM/PM will be reversed.

Line 220 - 240 these lines input the time in hours (h$), minutes (M$) and seconds (S$). Again we have a length trap ensuring no more than two characters.

Line 250 is the end of the subroutine and control will jump back to line 30

Line 30 here we concentate the H$,M$, and S$ into TI$ which immediately starts the clock going.

Line 40 another subroutine call to line 300, saving return to line 50.

Line 300 a simple print statement, "do you want the alarm on?" Y or N

Line 310 -330 the 'guts' of the GET statement.

Line 340 "N" was pressed so return with the variable AL = 0

line 350 "Y" so we call subroutine at line 220 which is the input routine called earlier omitting the AM/PM part of it.

Line 360 here we concentate H$, M$, and S$ into AL$ and set AL = 1 (alarm on)

Line 370 returns to line 50.

Line 50 One more subroutine call to line 500 which simply print a tabulated output to the screen - the clock's box!

Line 100 This is the start of the main processing loop which ends at line 180 where it is directed back to line 100. This particular line is testing if the hours part of TI$ has reached 13 and if it has, we don't want a 24 hour clock so we call the subroutine at 400 which will reverse the AM/PM display. Look at lines 400 to 430.

Line 110 -130 we split the HHMMSS of TI$ into separate parts H1$, M1$ and S1$

Line 140 We can now print these strings into the clock's box

Line 150 this is an interesting line the first part IF AL THEN - IF AL = 0 then AL is false and so the rest of the line is NOT executed. If AL is NOT zero, then it is true and so the next IF...THEN is executed, which simply compares TI$ with AL$. In other words is it time to sound the alarm? If they are equal then AL is incremented to 2

Line 160 tests if AL equals 2 and if it does it calls the subroutine at line 450. If we look at line 450/460 we see that border is poked with the value held in X, and then X is incremented. A test on X keeps it between 0 and 15. A loop was not employed here otherwise things elsewhere would slow down whilst the loop was executed. So, just one colour increment and end the routine with a return to line 170.

Line 170 another GET key statement. Notice the IF...THEN tests to find a string which contains a character. If it is a null string no further action is taken otherwise AL is made to equal to zero, in other words the alarm is switched off. The final part of the line sets the border colour to black just in case the alarm was turned off during another colour in the cycle.

Line 180 transfers control back to 100 to continue with the main loop.

And there you have it. I suggest you study the listing until you are sure you understand it thoroughly. I used the border colour changes for the alarm because we haven't yet dealt with soundfx - that is to come quite soon.

Well, that's it for this issue, but more exciting programming is coming your way next month. We shall be looking at HIGH SCORES and how to keep them in order, how to sort tables and very much more.

Until then, happy programming..

I.D.O.S. 2

CDU's Interactive Disk Operating System gets an uplift

RICHARD DAY

How many DOS programs have even published in recent years, I can name at least three that have been published in this magazine alone. All of them claim to allow the user to use his/her disk drive to it's absolute limit. Well, I've produced ANOTHER DOS program! Before you turn away, just read on, you'll probably find that this one is very different. For a start, the whole system is accessed through an editor (much like a monitor), BUT, all the commands are available through Basic. The editor also accessible from Basic AND from pressing ONE key only! Secondly, the editor recognizes any number (up to four) of drives and allows any one to be selected easily. If you're still not impressed, then try this, there is in excess of 66 commands recognized (not bad eh?).

Other features include the ability to:-

A) Print a text file to the screen
B) Write a text file to disk
C) Hex dump a file to screen
D) Hex code a file to disk
E) Disassemble a file to screen;
F) Assemble a file to disk;
G) List a Basic program to screen;
H) Write a Basic program to disk;
I) Change the name of a disk;
J) Recover scratched files;
K) Lock/unlock file(s) (including wildcards!);
L) Change the load address of a file;
M) Define up to 16 function keys;

and many more!

WHY A DOS PROGRAM?

The original program was written to simulate (emulate?) the ADFS commands on the BBC which are simple to say the least. To delete a file on ADFS you only need type:-

***DELETE <filename>**

to do the same on the trusty C64, the following command-essay must be typed

OPEN15,8,15,"S0:filename":CLOSE15

I ask you, which is the simplest?

The other reason I wrote this DOS program was for a bit of a challenge, before writing this program I had not dabbled in machine code disk programming at all, so I dived straight in....To complete the challenge, I tried to emulate the Amiga's CLI (which was it's original name). Why did I call it IDOS? That's easy:- I considered that the program was to be used from Basic and immediate mode so it had to be interactive, hence the name INTERACTIVE DISK OPERATING SYSTEM.

THE COMMANDS

I better stop babbling on or I lose half the readership (that would leave about 7 people reading this). IDOS can be accessed by pressing one key only, that being the RESTORE key. Once this is pressed, you are into the IDOS interpreter and commands can be entered by typing them in and pressing return. If the RUN/STOP key is held down while the RESTORE key is pressed then a normal warm start occurs. In the following, the word in capitals is the command and the rest are the parameters.
<addr> dictates an hex address (eg C000, 0801 etc.), this can be up to 4 characters in length.
<filename> is a filename (never!), this may be up to 16 chars long, do not include @0:.... or 0:.... as this is handled automatically. Wildcards can be used, but if no filename is given, * is used. Any spaces in filenames must be replaced by shifted-spaces (CHR$(160)) as a space is recognized as a parameter separator (any shifted-spaces found will be converted to normal spaces when the filename is extracted from the command), however, a filename can be enclosed in quotes (either ' or "), this means that normal spaces can be used in the name.
{ and } donate optional parameters, these may be entered or left out. It is important to note that if the interpreter (IDOS's) is expecting a parameter and it finds one, it will use regardless of whether that the parameter is the correct one (eg an <addr> may be used as filename or vice versa, for dummy <addr>'s use 0).
Each command can be abbreviated, this is usually the first two letters then a period (full stop). Commands can contain a wildcard (the character ?), this particular letter is ignored, eg ?ELP can be used for HELP or C?T can be used for CAT.
If you are unsure of the syntax of a command, type HELP and a list of the commands along with their syntaxes will be printed up.
I think that now I should describe the main operation of the program. After each disk operation, the error status of the disk is read and printed eg 00, OK,00,00). A prefix is used, the first number is the number of the current drive

being used, these are numbered 0 to 3 (see later). The second number (after the hyphen) is the device that you are using.
If at any time there is an error in the disk communication then IDOS beeps and flashes the screen.So 3-9:00, OK,00,00 means that drive 3 has been defined as device 9 (see later for how to do this) and the error channel is clear (ok). Any display routine (such as DUMP or DISS) can be frozen by any of the CTRL/SHIFT/COMMODORE keys and can be stopped by RUN/STOP.Whenever IDOS is accessed (either by the RESTORE key or the main SYS call) then it finds any available drives and enables them for use (so if you have 4 drives connected, you can use each one). What it actually does is to look through the drive table (ie the devices assigned to each drive number) and checks to see if that device is connected, if it is then it will read the error channel and print it. Once all the devices have been tested, IDOS automatically selects the device which has the lowest drive number.
NOTE that 'device' refers to a disk drive whereas 'drive' refers to a particular device, a particular device number being assigned to a particular drive number. To find which device is assigned to which drive type DEVICE <return>.

COMMAND EXPLANATIONS

**LOAD <filename> {<addr>} {N}
(abbreviation:L.)**

Loads the prg file <filename> into memory from <addr> on (if no <addr> or 0 is given, then the header address of the file is used as the start address). If the program is loaded into Basic memory (ie if the <addr> given is the start of Basic or no <addr> is given, or 0 is used) then the end of Basic is set at the end address of the load (ie if the file loaded is Basic then it is ready to run, as the normal load. But if the file loaded is code or binary then the Basic memory is left untouched, unlike the normal load).
This command does not use the kernal load file routine, instead, it uses it's own, which shows how many blocks have been loaded as the load proceeds. It also shows the start and end address of the load.
If the optional N is present then the file will be loaded using the normal Kernal Load routine. (this is automatically done if any part of the file which is loaded overlaps IDOS).

```
Examples:LOAD PACMAN
         LOAD 'ZAPPER 24' C000
         LOAD 'KILLER' N
```

**VERIFY <filename> {<addr>}
(abbr:V.)**

This verifies a file in accordance with LOAD (same parameters and syntax).
This again, does not use the kernal routine and actually counts the number of mismatched bytes (and shows this automatically at the end). Every 256 mismatched bytes IDOS will beep and flash the screen.

```
Example:VERIFY ZAPPER
```

**SAVE <filename> {L} {<addr>} {<addr>} {<addr>}
(abbr:S.)**

This saves a section of memory with name <filename> from <addr> (or Basic start address if not given or 0) to <addr> (or Basic end address if not given or 0) with header address <addr> (or 1st <addr> if not given or 0) with <addr>'s respective. Phew!This means you can save the memory from $C000 to $CFFF to disk so it will load in at say, $5000 (very useful when using sprite data or graphics memory).
IDOS automatically prefixes the filename with @0: for the save hence automatically using the save and replace facility of the disk drive. There is a lot of worry in Commodore circles about the safety of this facility so you can switch it off using the REPLACEOFF command and so IDOS will prefix the filename with 0: instead.
The optional L is a flag to signify the locking of the saved file, if this is present then the file is automatically locked once saved.

```
Examples:SAVE IDOS L 7000 9FFF
         SAVE MONITOR C000 CFFF
         SAVE 'ROCKET' L
         SAVE SPRITEDATA 4000 4FFF 2000
```

**APPEND <filename> {<addr>}
(abbr:A.)**

This uses the load routine to append the named file on to end of the current Basic program. The <addr> is ignored.

```
Example:APPEND RENUM.ROUT
```

**EXEC <filename> {<addr>} {<addr>}
RUN <filename> {<addr>} {<addr>}
BOOT <filename> {<addr>} {<addr>}
(abbrs:E.,R.,B.)**

This loads and executes a file. The filename and first <addr> are as for LOAD. The second <addr> is an optional execution address (for machine code programs only). If this is not present then the file is run either as a Basic program (as the Basic RUN command) or called as a machine code program from the load address.

This means that typing BOOT <return> will load and run the first program on the disk whether it is Basic or machine code (very clever!).

It is important to note that IDOS automatically switches off the function keys before executing the program, so that if your program interferes with the memory that IDOS occupies then the machine won't crash.

```
Examples:BOOT TURBO
         EXEC SAVER
```

**COMMAND <command>
(abbr:C.)**

This executes a normal DOS command (eg R0:PROG1=PROG).

Example:COMMAND R0:ZAPPER1=ZAPPER

CAT {<filename> {<filename>...}}
DIR {<filename> {<filename>...}}
(abbrs:CA.,D.)

These two commands, surprisingly, read the disk's directory, the <filename>'s are optional masks ie CAT *=PRG will show entries of only PRG files, CAT IDOS will show the entry for IDOS only, CAT ID* will show all entries whose filenames begin with 'ID'. If more than one filename is used, then if the filename fulfils either mask then it is printed, eg CAT IDOS PROG1 PROG2 will list any entries for IDOS or PROG1 or PROG2.

Example:CAT IDOS* HIRES*

DUMP <filename> {<addr>}
PRINT <filename>
TYPE <filename>
LIST <filename> {<linenumber>}
DISS <filename> {<addr>}
(abbrs:DU.,P.,T.,LI.,DIS.)

These commands display a file (any type) in a number of ways:-

DUMP shows a hex and ascii dump of the file, the <addr> is optional and is from where the dump starts (ie if a file would load in at C000 and C300 is used as the <addr> then the first $300 (=768) bytes are ignored).

PRINT prints a file (to the screen), this routine prints only printable characters.

TYPE is the same as PRINT but prints all characters.

LIST displays a file as a Basic program (ie lists it), the optional linenumber is where the list is to commence (as with the <addr> in the DUMP command).

DISS interprets the file as a machine code program and disassembles it, the display shows the address, the bytes as hex, ascii then as 6502 mnemonics.

```
Examples:DUMP IDOS
         DISS 'ROM80 CODE'
         TYPE 'ID.TEXT'
         LIST ORRERY
         DISS IDOS 8000
```

WRITE <filename> {L} {<ft>} {<addr>}
CODE <filename> {L} {<ft>} {<addr>}
ASSEM <filename> {L} {<ft>} {<addr>}
PROGRAM <filename> {L} {<ft>} {<linenumber> {<step>}}
(abbrs:W.,COD.,AS.,PRO.)

These four commands allow a file to written to the disk in a number of forms (they are actually the direct opposite of the previous commands). The optional L if present, will lock the file once you have finished entering it. The <ft> is the filetype of the file and is optional, this may be P, S, U or D (PRG, SEQ, USR or DEL, yes, you can have a file which shows up as DEL on the directory and can be used). If no <ft> is given then the default is PRG. The <addr> is the start address which is where the file will load back. This <addr> is the first thing that is written to the disk, if no <addr> is given, then no address is written to the disk.

WRITE writes a text file to disk (NOT necessarily a SEQ file). Type in up to 40 chars of text then press return, continue this until you've finished, then press RUN/STOP, the file will be closed and you will be returned to the normal interpreter. During the typing in stages, there is no prompt printed, only a cursor.

Example:WRITE TEXT.FILE S

CODE writes a file to disk from lines of hex data, the address will be printed and then you must type the data in (there may be as many bytes as you like up to two lines of data), to enter ascii use ' then the ascii chars, all the text from then on until a space is met is taken as being ascii. This means you cannot enter a space as ascii, you have to use hex 20 (=32 in decimal - the ascii of space). To end entry, press return on a line with only an address on.

Example:CODE SPRITEDATA L 4000

Then IDOS will prompt with:-

```
4000
```

Then type in the hex numbers:-

```
4000 20 50 40 23 4F 'HA 20 'HA
```

When finished, press return on a address:-

```
432F (press return now)
```

ASSEM writes a file to disk from lines of 6502 assembly, the address is printed then you must type in a mnemonic then any operand, then press return. Numbers (addresses or bytes like in JSR $2000 or LDA #$20) can be entered as hex (using $), decimal (no prefix), ascii (using ', only applicable to byte operands) or binary (using %). To end entry, press return on a line with only an address, as CODE. Data can be entered using BYT (bytes) or WOR (words).

Example:ASSEM CODE L C000

```
C000 LDA #0
C002 STA $D020
C005 LDA 53265
C008 AND #%11110111
C00A STA 53265
C00D LDA #$20
C00F LDY #$C0
C011 JMP $AB1E
C014 BYT 147,$0D,$0D,"HI, THIS IS A
```

```
DEMO!",$0D,$0D,0
C02D WOR $C020,$AB1E
C031 (press return now to end)
```

NOTE:IDOS automatically calculates and prints the next address.

PROGRAM writes a file to disk as a Basic program, the <linenumber> is the first linenumber of the program, the step is the step for the linenumbers, the default for both of these is 10. The linenumber is printed up then a line of Basic must be entered (normal rules apply, Basic abbreviations can be used), the return must be pressed. Then the step is added to the previous linenumber and this is used as the next linenumber (incidentally, the current linenumber can be changed at any time during entry by going to the start of the line and entering a new linenumber) . To end entry, press return on a line with only a linenumber.

Example:PROGRAM 'HEX CALCULATOR'

```
10 POKE53280,10 :POKE53281,2 :?"{CLR} {CTRL-7}"
20 INPUT"{DOWN}ENTER HEX NUMBER";H$
30 GOSUB100:?"{DOWN}$"H$"="DC:GOTO20
40 (at this point, type 100 over 40)
100          DC=0:FORI=1TOLEN(H$):A$=MID$
(H$,I,1):A=ASC(A$)-48:A=A+7*(A>9)
110 DC=DC*16+A:NEXT:RETURN
120 (press return now to end)
```

SIZE <filename>
(abbr: SI.)

This shows the size of any file and where it would normally load into memory, it is essentially the same as LOAD but does not enter the bytes into memory.

Example:SIZE 'IDOS.SOR'

RENAME <newname> <oldname>
(abbr:RE.)

This renames a file from <oldname> to <newname>, just like the R0: DOS command.

Examples:RENAME IDOSL IDOS
RENAME 'ROM80 PROG' 'ROM80 PROG1'

COPY <newname> <oldname> {<addname> <addname>...}
(abbr:COP.)

This copies the file <oldname> onto the same disk with name <newname>. Any <addname>s are appended onto the end of the new file.

Examples:COPY 'BACKUP' 'PROG'
COPY IDOSB IDOS
COPY TEXT TEXT1 TEXT2 TEXT3

COMPACT
VALIDATE
(abbrs:COMP.,VA.)

This validates the disk (as V0).

DELETE <filename> {<filename>...}
SCRATCH <filename> {<filename>...}
ERASE <filename> {<filename>...}
(abbrs:DE.,SC.,ER.)

These commands delete a file from the disk (as S0:) and allows you to delete any number of files, using wildcards or any number of filenames as long as the length of the command does not exceed 40 chars (1 line).

Example:DELETE IDOS

REMOVE <filename>
(abbr:REM.)

This command automatically unlocks a file then deletes it.

DISKNAME <diskname> {<ID>}
(abbr:DISK.)

This command changes the name and id of the current disk, <diskname> is a name of up to 16 chars (any spaces must be shifted-spaces, as in a filename).

Example:DISKNAME 'ASSEMBLER7' A7

NEW <diskname> {<id>}
FORMAT <diskname> {<id>}
(abbrs:N.,F.)

These commands format the current disk (as N0:), if no <id> is specified then the name of the disk is changed to <diskname> and the disk cleared of all files (this is unlike formatting because it does not construct the tracks and sectors as does the format when <id> is specified). NOTE:There is NO 'Are you sure (Y/N)' message as a safety net, once return is pressed the process is unstoppable.

Example:FORMAT ASSEMBLER8 A8

INIT
(abbr:I.)

This initialises the current disk (as I0). This routine is only useful to check to see if there is a disk in the drive (an error will occur if there isn't).

DERR
ERROR
(abbrs:DER.,ERR.)

These commands read and print the disk status, like OPEN15,8,15 :INPUT#15,ER,ER$,TR,SE :CLOSE15 :PRINT ER;ER$;TR;SE. Since this is automatically done at the end of each disk operation, these routines are only present for convenience. (Though they can be used in Basic programs to save bother).

PROTECT <filename>
LOCK <filename>
(abbrs:PROT.,LOC.)

These commands protect a file from erasure, a protected file cannot be erased at all, unless it is unlocked or the program is resaved. Any protected files show up with a 'less-than' (<) sign immediately after it's filetype in the directory. Wildcards can be used in the filename and so more than one file can be locked in one go.

Example:LOCK IDOS*

UNPROTECT <filename>
UNLOCK <filename>
(abbrs:U.,UNL.)

These commands reverse the previous commands and make files erasable.

Example:UNLOCK IDOS

RECOVER <filename> {L} {<ft>}
(abbr:REC.)

This command recovers the directory entry for a file hence recovering the file after deleting. However, the sectors are not re-allocated, there you should validate the disk after this command.

The default <ft> is PRG so if this is not present the file will be marked as PRG. The optional L is for the automatic locking facility as with the SAVE command.

Example:RECOVER UNDELETER L

DESTROY <filename>
(abbr:DES.)

This command wipes over the entry for a filename in the directory, hence effectively deleting the file. However, the program is still on the disk and the sectors are still marked as used, therefore you should validate the disk afterwards. This command is only of use if the normal delete command will not work.

ADDRESS <filename> {<addr>}
(abbr:AD.)

This command changes the load address of a file (the first two bytes of the file) to that of <addr> (Basic address if not present).

Example:ADDRESS SPRITEDATA 6000

DRIVE <drive>
(abbr:DR.)

This sets the current device to that set for the specified <drive> in the device table (see below). <drive> is numbered from 0 to 3, any other will give an error.

Example:DRIVE 2

DEVICE <drive> <device> {<drive> <drive>...}
(abbr:DEV.)

This sets the device for a particular drive number. The command prints up the available devices from the current table along with there respective drive numbers.

Examples: DEVICE
DEVICE 0 8 1 9 2 10 3 11
DEVICE 0 9 1 8

This command is useful if you have two drives, of which one is a 1541 and the other is, say, an Excelerator drive (what happened to them anyway?). The 1541 requires major surgery to alter it's device number, but the Excelerator has DIP switches which make this a doddle. Therefore, normally, the 1541 would be device 8 and the Excelerator device 9. This means that any other DOS program would use the 1541 as default. However, with IDOS, you can set up so drive 0 is device 9 (the Excelerator) and drive 1 is device 8 (the 1541) with DEVICE 0 9 1 8. Now, IDOS will select the device with the lowest DRIVE NUMBER, which in this case is device 9, the Excelerator. So IDOS would default to the Excelerator, which makes life a lot easier!

HEX <decnum>
(abbr:H.)

This prints up the hexadecimal equivalent of <decnum>, range 0 - 65535 (useful for working out addresses for <addr>s).

Example:HEX 64738
would give:- =$FCE2

DEC <hexnum>
(abbr:DEC)

This prints up the decimal equivalent of <hexnum> (range 0 - FFFF).

Example:DEC C342
would give:-
=49986

CALL <addr>
(abbr:CAL.)

This calls a machine code routine stating at <addr> (similar to SYS but uses a hex address and stays within IDOS).

Example:CALL C342

RESET
(abbr:RES.)

This command reset the computer (as SYS 64738).

BASIC {<addr>}
(abbr:BA.)

This command sets the start of Basic to <addr> and reset Basic pointers, if no <addr> is specified then the command just reset the pointers (as NEW - the Basic command).

Example:BASIC 200COLOUR <bord-col> <scrn-col> <text-col>
(abbr:COL.)

This command sets the colours for use in IDOS, whenever IDOS is entered then the current colours will change to these specified. Each is a number and ranges from 0 to 15 (decimal).

Example:COLOUR 6 15 6

PROMPT <prompt-text>
(abbr:PROM.)

This sets the text which is printed before each command is entered (the prompt), any amount of text (up to 80 chars) can be used. If a carriage return is required, use the left-arrow (top left of the keyboard).

Example:PROMPT ENTER COMMAND

FKEY {<key-no> <m><f-text><m>}
(abbr:FK.)

This defines the text for one of the 16 possible function key definitions, <key-no> ranges from 1 to 16, the <m> is a marker character, this marks the start and the end of the definition but is not included in the definition, this allows spaces to be entered into definitions. If returns are required in the definition, again use the LEFT-ARROW. If FKEY is entered on it's own then the definitions of all the function keys are printed along with the key combination to achieve that definition.

Examples:	FKEY 1 'CAT (left-arrow)'
	FKEY 3 'LOAD '

The function keys are defined as follows:-

KEY 1	{F1}:'CAT (left-arrow)'
KEY 2	{SH+F1}:'REPEAT'
KEY 3	{F3}:'LOAD '
KEY 4	{SH+F3}:'FKEY '
KEY 5	{F5}:'SAVE '
KEY 6	{SH+F5}:'HELP (left-arrow)'
KEY 7	{F7}:'BOOT '
KEY 8	{SH+F7}:'QUIT (left-arrow)'
KEY 9	{CM+F1}:'TYPE '
KEY10	{CT+F1}:'WRITE '
KEY11	{CM+F3}:'DUMP '
KEY12	{CT+F3}:'CODE '
KEY13	{CM+F5}:'DISS '
KEY14	{CT+F5}:'ASSEM '
KEY15	{CM+F7}:'LIST '
KEY16	{CT+F7}:'PROGRAM '

NOTE:SH means SHIFT, CM means COMMODORE KEY and CT means CTRL.

FUNC <key-no>
(abbr:FU.)

This command executes the function key definition <key-no> without the need to press the keys.

KEYON
KEYOFF
ERRON
ERROFF
READON
READOFF
REPLACEON
REPLACEOFF
(abbrs:K.,KEYOF.,ERRON,ERROF.,REA., READOF.,REP.,REPLACEOF.)

This commands switch on or off a particular 'parameter' of IDOS.

KEY governs the function keys,

ERR governs the printing of the error status of the disk after a disk operation,

READ governs the reading of the start address of a file when using the DUMP, PRINT, TYPE, DISS etc. If **READOFF** has been executed then when the file is read then it's start address is 0000 and the first two bytes (which would be the start address) are treated as part of the program (displayed as so).

REPLACE governs the save & replace feature when saving (ie the using of @0:filename).

REPEAT
(abbr:REPE.)

This command repeats the last command executed.

EXIT
QUIT
(abbrs:EX.,Q.)

These command allow you to leave IDOS and return to Basic, the function keys being left in there state (ie on if they were on and off it they were off).

HELP
(abbr:HEL.)

This displays all the commands along with their syntaxes, the display can be paused by CBM/SHIFT/CTRL and stopped by RUN/STOP.

KILL
(abbr:KI.)

This kills IDOS: it switches off the f-keys, leaves IDOS and makes it impossible to re-enter IDOS through the use of the RESTORE key. IDOS can be still accessed by the SYS, which re-enables IDOS to be accessed by the RESTORE key.

ON THE DISK

PHEW!! That's all the commands, quite a few isn't there? You will notice that some commands do the same thing as others, this is just for convenience, it may be easier to remember ERASE rather than SCRATCH or CAT instead of DIR.

THE SYS COMMANDS

I thought now I would explain the SYS commands:-

SYS32777,"<command>..." - this executes the command in the quotes eg:

SYS32777,"DELETE PROG1" - will delete 'Prog1' from the disk and display the disk status.

**SYS32777,A$** - is perfectly legal and in this case the text contained within variable A$ will be treated as the command.

So a program could be written like this:-

```
10 INPUT"COMMAND";A$
20 SYS32777,A$:GOTO10
```

would imitate the IDOS interpreter.

SYS32780 and SYS28672 enter IDOS (like pressing the RESTORE key).

SYS32783 enables IDOS to be accessed by the RESTORE key.

SYS32786 disables access by RESTORE.

Location 700 always holds the device currently in use, whereas location 701 holds the error number of the last disk operation (0=OK).

That's it there's no more!

SAVER PROGRAM

Included on the disk is a program called 'Saver', this allows you to save a copy of IDOS to all of your disk with ease. When run, you will be asked to insert a disk then press a key. Put your disk in the drive and then press a key, IDOS will be automatically saved and locked on that disk. Have a look at the program, you'll see how nice and ordered it looks. Using IDOS from Basic means that programs can be streamlined and decipherable. Well, I must sign off now, I hope you find IDOS as useful as I did.

TIMEWORKS SOFTWARE SALE

	RRP	OFFER
PARTNER 128 (accessory pack cartridge)	69.95	24.95
PARTNER 64 (accessory pack cartridge)	59.95	19.95
SWIFTCALC 128 (spreadsheet with Sideways)	69.95	24.95
SWIFTCALC 64 (spreadsheet with Sideways)	49.95	19.95
WORDWRITER 128 (word processor with spell checker)	69.95	24.95
WORDWRITER 5/ 64 (word processor with spell checker)	39.95	29.95
SIDEWAYS 64/128 (Sideways print utility)	29.95	9.95

VALUETIME PRODUCTS

**These C64 disk software products normally retail at £9.95.
Offer price £4.95 each or any 3 for £12.**

The Musician (3 different programs to compose and play music)
The Artist (3 programs to create multicolour graphics, print your artwork on most popular printers, design sprites)
Disk Drive Manager (load your programs up to 5 times faster, add new disk commands, backup your disks)
The Entertainer (3 fast action arcade games)
The Entertainer 2 (3 challenging strategy games)
The Entertainer 3 (3 fast-paced action/adventure games)
The Educator (improve your typing skills, learn algebra and improve your numeric skills by playing games)
The Educator 2 (for children aged 5-10. An entertaining way to learn to tell the time, spell and use the traffic lights)
Electronic Cheque Book (organise, classify and record your chequebook transactions)
The Home Banker (5 programs to help with personal finance)
The Home Manager (Wordprocessor, Database, Calculator and Memo Pad)
ValueCalc (Spreadsheet) *(very limited stock)*

DTBS, 18 Norwich Avenue, Rochdale, Lancs. OL11 5JZ
Tel: 0706 524304
Send SAE for descriptive list

VISA

Note: DT Books and Software (DTBS) is the new trading name of Adamsoft.

STRATEGY ADVENTURE

C64 disks only

INFOCOM	
BALLYHOO	£14.95
BUREAU CRAZY C128	£14.95
HITCHIKERS GUIDE	£9.95
LEATHER GODDESS	£9.95
INTERSTEL	
EMPIRE	£29.95
LUCASFILM	
ZAK MC KRACKEN	£14.95
MICROLEAGUE	
MICROLEAGUE BASEBALL	£24.95
MICROLEAGUE FOOTBALL	£24.95
MICROLEAGUE WRESTLING	£19.95
SSI	
50 MISSION CRUSH	£12.95
BATTLES OF NAPOLEON	£24.95
BUCK ROGERS	£24.95
CHAMPIONS OF KRYNN	£24.95
CURSE OF AZURE BONDS	£24.95
DRAGON STRIKE	£24.95
FORTRESS	£12.95
GEOPOLITIQUE 1990	£12.95
HILLS FAR	£19.95
OVERRUN	£24.95
PANZER STRIKE	£24.95
PHANTASIE III	£24.95
POOL OF RADIANCE	£24.95
PRESIDENT ELECT	£14.95
QUESTRON I	£19.95
QUESTRON II	£19.95
ROADWAR EUROPA	£19.95
SECRET OF SILVERBLADES	£24.95
SIX-GUN SHOOTOUT	£12.95
STORM ACROSS EUROPE	£24.95
TYPHOON OF STEEL	£24.95
WAR OF THE LANCE	£24.95
WARGAME CONSTR SET	£19.95
SUBLOGIC	
FLIGHT SIMULATOR II	£24.95
NIGHTMISSION PINBALL	£12.95
STEALTH MISSION	£24.95
TELARIUM	
DRAGON WORLD	£9.95
WIZARD	
SUPERSTAR ICE HOCKEY	£14.95

Hint Books £7.95 Each:
BARDS TALE I, II or III, BUCK ROGERS, CHAMPIONS OF KRYNN, CHAOS STRIKES BACK, CURSE OF AZURE BONDS, DRAGON WARS, DRAKKHEN, DUNGEON MASTER, ELITE, ELVIRA, INDIANA JONES L.C. ADV, MANIAC MANSION, MIGHT & MAGIC I OR II, NEUROMANCER, POOL OF RADIANCE, SECRET OF SILVER BLADES, STAR FLIGHT, ULTIMA III, IV, V or VI, WASTELAND, ZAK MC KRACKEN.

Mail order only. Please allow 28 days for delivery.
Please make cheques and postal orders payable to CINTRONICS LTD.
Free post and packaging within the UK. Europe add £2 per item. Overseas add £4 per item.

**CINTRONICS LTD.
16 Connaught Street,
London W2 2AG**

TECHNO-INFO

Our resident guru, JASON FINCH, answers more of your perplexing problems

It's TECHNO-INFO time again and this month there are plenty of letters to keep you busy. I hope that many of you, and in particular the people who wrote the letters, will find my replies both informative and useful. The regular UPDATE section is here as well, this time letting you know how to correct a rounding error in the CHEQUE BOOK program by PETER WEIGHILL. Of course it is now June and we are all ready for a new Techno-Info Challenge to be set. In case you missed the March issue, this is a brand new (or at least it was in March!) quarterly feature of Techno-Info that will allow you to play about to your heart's content at trying to program whatever I ask of you. All Challenges will be in BASIC so no machine code knowledge is required. There are details of this month's Challenge after the letters, together with details of last time's winning entries! So then, let us get on with the proceedings.

TAPE TO DISK

Dear CDU,
I have recently purchased a disk drive for my Commodore 64 and I would like to know if there is a way of converting material on tape onto disk. I would also like to know of a suitable database, so that I can catalogue my collection of football programs.
S.Miller, Scunthorpe.

Dear Mr.Miller,
From your letter, I presume that you mean commercial software. The way to convert non-commercial material to disk is to just load it from tape and then enter SAVE"progname",8. However, for games and the like, it is a matter of purchasing a backup cartridge (such as Action Replay from DATEL or Super Snapshot from FSSL). There is a lot of controversy regarding the use of these but it is the only way that you will do it. It is a simple matter of loading your game from tape, pressing a button on this plug-in box of tricks, and then entering a suitable filename for the disk version. When it comes to databases, there are lots that you can choose from. If you want a very advanced database you can expect to pay around £40. Ones like this, such as SUPERBASE, can be obtained from a company called FSSL. Their catalogue was published in the March 1991 issue of CDU and their address can be found elsewhere in this section. However, you could buy a back issue of this magazine, one in which a database was published of course. That will be a great deal cheaper and they should all cover your needs. Look in the back issues section of one of the magazines and see what program seems to suit your needs.

SERVICE MANUALS

Dear CDU,
Thanks very much for the assistance with my problem with quotients and remainders. There is no doubt in my mind that your explanation on the method of finding them should be included in every Commodore Systems Guide! I have read the section they include on the INT function again and there is nary a word about moving the bits of the formula around as you did. In a recent issue of CDU you mentioned a leaflet on degunging a keyboard on a C64. I have two C128s and I do not suppose that either of them will get into the state the leaflet talks about but I would like to get the top off both of them sometime to blow out the accumulated dust and whatever. I am told that you have to undo the screws in the base and press in the sides of the lower part to clear the plastic clips. I have not tried it but it sounds reasonable. It is a pity that there is not a book I can purchase - similar to a car service manual - that demonstrates to a novice like me the way to dust out my computers, disk-drives and printers! Thanks once again for your precise explanations.
Eric Frost, West Sussex.

Dear Eric,
I am glad that you found my explanations clear and useful. The way to get into the C128 is exactly as you have been told - unscrew five little screws and release the cover from a couple of clips. However, when it comes to service manuals, I'm afraid that you are wrong. There are in fact such publications around which you will be glad to hear are now available to the general public. They tend to cover the more technical side of things, but Commodore Service Manuals for most Commodore equipment can be purchased from FSSL. The address is Masons Ryde, Defford Road, Pershore, Worcestershire, WR10 1AZ and the number is 0386-553153. They should be able to provide you with what you want.

GRAPHICS CONVERTER

Dear CDU,
I would be grateful if you could advise me of any routine, PD or commercial program that would enable me to convert a multicolour screen (eg. Koala) to a high resolution screen (eg. Doodle) and save the converted file to disk. There are plenty of PD multicolour screen files available and I would like to convert them for use in GEOS. However, I first need them to be converted to Hi-Res format before I can convert them to GEOS. I should be grateful of any help that you are able to give. I am sure that this information will be of use to many readers interested in graphics packages and GEOS. There is a converter routine in ICON FACTORY but I find that this does not do a very good conversion. Keep up the good work with CDU.
Michael Pearson, Stratford-upon-Avon.

Dear Michael,
Unfortunately the conversion obtained from Icon Factory is probably going to be the best that you will achieve. I know of plenty of excellent packages that do the conversion well from Hi-Res to multicolour but not the other way around. You see, in multicolour mode you can have four colours in each "square" on the screen, but with high resolution you can only have two, one foreground and one background. Therefore it is hard to convert the pictures. A decision has to be taken as to which of the four colours are most important and can be used to best effect in the high resolution picture. If it was possible to make a high resolution picture look as good as a multicolour one, no-one would bother loosing the resolution given by Hi-Res. Therefore, the converter that you already have is the only one that I could recommend.

VIEWS ON PIRACY

Dear CDU,
I would like to put forward my views on what I know is a very controversial subject; that of backup cartridges and their use for pirating software. All adverts for these cartridges say "XXX does not authorise the use of this product for copying copyright material". Some, however, say "freeze and reload programs at super fast speed". I would like to hear from any owner of a backup cartridge who has not used it to hack a game or program even for their own use with a turbo-loader (it is illegal to copy programs even for your own use). Will there ever be any law passed to control the supply of these piracy-encouraging utilities? Piracy leads to high software prices because authors allow for about five pirate copies per game, so they set the price five times higher. So when you next copy your friend's latest game, pause and think about the high prices that you are contributing towards.
P.T.Lenfestey, Guernsey.

Dear Mr.Lenfestey,
Thanks for your letter. Techno-Info is here to allow readers to say what they think - not only to help with your problems, remember. I agree that most people will have used such devices for the "wrong" purpose but so long as the cartridges have other features, and not only the facility to copy software, then they cannot be banned. In a previous reply in this section I have told someone to use a copier to transfer their games from tape to disk. I, personally, don't think that it is fair to make someone buy a disk version of their game when they have already bought the tape version when they didn't have a drive. Also, I think that piracy and high prices cause a vicious circle to be created. Piracy leads to higher prices, yes, but higher prices lead to yet more piracy. And so the process continues. I am not saying that I condone piracy, I am just saying some simple facts. Put it like this, if there was a fully featured word-processor on sale for fifty pounds, and someone wanted one but couldn't pay that much they would illegally make a copy from a friend's perhaps. If that same word-processor was available for a tenner, that wouldn't happen would it. This ties in with your quote about making packages five times more expensive. But having said a few controversial things in one paragraph (which are my views and not necessarily those of the magazine or other Editors) I will end by asking a couple of questions myself: How come there are budget games for only £1.99, and how many of them are illegally copied? The answer to the last question is very few because people don't mind paying two quid as opposed to ten. If anyone wants to continue this "debate" then please feel free to write to the Techno-Info address with your views.

HARD DRIVES

Dear CDU,
I am writing to see if you can tell me where I can get a hard disk drive for the Commodore 64. I know it is possible because I have seen mention of it in the Gateway package which is available from FSSL. You published its catalogue in March. Also can you tell me how much it will cost and how to configure it.
R.S.Pope, Bath.

Dear Mr.Pope,
Although the hard drives weren't mentioned in the catalogue in CDU, the company that you mention, FSSL, will also be able to supply you with a hard drive. They are probably the only place in Britain from where you could obtain such advanced hardware. At the time of typing this, the price is £499.95 for a 20 Megabyte CMD HD series hard drive (to give it the full title!), and for a 40 Megabyte one you can expect to pay about two hundred pounds more. However, before ordering, if in fact you do still want one!, I would recommend that you telephone the sales department on 0386-553153 to check availability and prices. I presume that details on configuration and connecting it to the computer will be given to you when you buy one.

RE-ALIGNMENT FOR YOU

Dear CDU,
I have two 1541 disk drives, one of which is a good deal older than the other. Programs saved on disks which have been formatted on one drive will not load on the other.

Most commercial software will not load on the older drive. Can you explain this please and suggest how it may be corrected? - if such a thing is possible.
A.P.Forrest, Merseyside.

Dear Mr.Forrest,
My diagnosis of the symptoms is that your older drive needs realigning. Over a period of time, like many other things, the moving parts tend to go out of line. With a disk drive, the "head" and everything else that controls the reading and writing of information become very gradually misaligned. The good old Datasette units are similar. The older drive formats disks wrongly and saves them incorrectly in comparison to other drives. That is why nothing saved on the older drive will load on the new drive. Also, the drive cannot read the commercial software properly and that is why that won't load. But the problem can be corrected - at a small cost. You can buy a DRIVE ALIGNMENT package from FSSL (address under reply to SERVICE MANUALS above) which will cost £24.95 plus £1.45 p&p. It consists of an easy to understand manual together with some software. This software, it is claimed, will load when nothing else will. The package includes a drive speed test to make sure that the drive head is rotating at the correct speed of about 300rpm. For a run-down on what is wrong and how to correct it, buy the package. However, you'll probably find that your programs saved by the mis-aligned drive will never be able to be loaded again once the drive is correctly aligned. If you are worried about this, load each program in and then just save it onto a different disk in the newer drive.

C128 + 1551 DD

Dear CDU,
Would you please help me with this problem. I am trying to use my 1551 disk drive (ex. plus/4) with my new (second hand!) C128. The connector is different to the socket. Is there an adaptor available? Will it work anyway and if not what disk drive should I get?
David Joberns, Essex.

Dear David,
I must confess that I am not sure! I am not familiar with the 1551 drive or the lead that is usually used to connect it to a Plus/4. I would suggest that you telephone the technical department of a company called Meedmore in Merseyside on 051-521-2202 who should be able to help with this particular type of problem. If it is possible to connect the two pieces of hardware then they will know how. As to whether the drive would work, if you managed to connect everything up then there is no foreseeable reason for it not to be possible to save programs to disk and to load them back again. However, because of the presumably different DOS types between the 1551 and a 1541 compatible it is unlikely that the majority of commercial software would work on the 1551. I would therefore recommend that instead of trying to connect the 1551 which could cause incompatibility problems, you should buy a 1541 compatible drive. These can be obtained from most good computer stores that stock Commodore 64 equipment. Such drives are the 1541 (with various suffixes such as -II and -C), the Oceanic drive and the 1571. The latter is in fact the drive that is "meant" for a C128. I hope that I have been of some help to you.

INTERFACES

Dear CDU,
Thank you for a very interesting and useful magazine, I haven't missed one from the start. Keep up the good work. The reason I am writing is that I was given a PET computer a few weeks ago, with it an 8250 disk drive. I do believe that this drive can be connected to a C64 using an IEEE488 interface. I would be very grateful if you could tell me where I can purchase this interface or if any reader might have one for sale. I would like to be able to connect this drive to the 64 for the extra storage it offers.
C.J.Thomas, Bristol.

Dear Mr.Thomas,
The interface that you mention should be available from Meedmore Distribution in Merseyside. They stock a huge range of such accessories and various leads and connectors. The place is a warehouse and not a shop and most ordering is done through the post. I suggest that you give them a call on 051-521-2202 to check the stock code and price. If you do eventually get the drive hooked up to the 64 then I presume that you are aware that commercial software designed for the 1541 disk drive will not work with the 8250.

CONFIGURATION PROBS

Dear CDU,
I read in January's issue about the problem that David Medland had with not being able to use the print option on the OCP Advanced Art Studio. I also have the same problem. No matter which way I configure the program I get the reply "Error during print - Device not present". I have tried every possible solution to it with no joy and I would be very grateful if you could help. I have a Commodore MPS1230 printer.
Paul Stanton, The British Forces.

Dear Paul,
David's problem was solved by just entering the basic standard responses to the configuring questions. Your problem sounds a little different. "Device not present" makes me think that the problem is the hardware and not necessarily the software. The printer may be set as device five or something, or there may be a daisychaining problem. Enter the following line in BASIC: OPEN 4,4: PRINT#4,"TEST": CLOSE 4. If the printer does nothing then the device number needs changing I would think, unless there is something more serious wrong with the printer. If that line did work then you must make sure that the OCP program knows that your printer is a serial one and device number four. If all that seems fine then I suggest you go through the configuration program step-by-step with the manual at your side and thoroughly check that each input is correct.

AN ODD BUG

Dear CDU,
Over the past four years that I have had my 64C I have become increasingly aware of a bug, which I think is in one of the computer's internal chips. The problem is not a terribly great one, but it is very annoying. It only seems to crop up when I am using my games software; which I tend to use 75% of the time at the moment; and depending on which game I am using determines how the program is affected by the bug. For instance if I have an arcade type game running such as Bruce Lee, Beach Head II, Turbo Outrun, etc. by US Gold, the bug seems to affect the cumulative element in the scoring of the game. When the tens, hundreds and thousands column is required to zero itself and advance the next column forward one, it seems incapable of achieving this. However, in games such as TV Sports Football and Sentinel, the bug completely destroys gameplay. As the bug recurs throughout my software collection I could only assume it is a hardware problem, and having only upgraded to disk in the last year, I came to the conclusion that the problem must be with the computer itself. So I recently purchased the C64 DOCTOR from FSSL and tested the internal chips, but they all passed the tests. So in desperation I am writing to you to see if anyone you know can advise me on what I should do, as I don't want to send my computer off to be repaired if it is something that nobody knows how to rectify.
Mark Fletcher, Glasgow.

Dear Mark,
This certainly is a very odd bug and I would agree with you that it must lie with the computer itself. It is unlikely to be the fact that all your games are corrupted in some way, but have you tried any of them on a different system? I have never heard of the problem before but I would suggest that to get it sorted out you are going to have to send it somewhere. I can't say that someone is going to know exactly what the fault is but there are a number of ways to narrow it down. It is not likely to be anything to do with the input/output chips, the sound chip, probably not the video interface chip and so on. I would think that the problem lies deeper, perhaps in the actual ROM or maybe there is some error in the circuitry. I should think that you would be best off sending the computer for a sort of "all inclusive and paid for" repair. For example you could telephone OASIS Computer Services on 0722-335061. They do a repair service for Commodore 64s with a one week turnaround costing thirty pounds. Perhaps offer to send them a couple of your games just to show them what is happening. They would possibly be able to sort it out or point you in another direction to a company that can repair your machine completely. I hope that I have started you on the right track at least.

FOREIGN 1571

Dear CDU,
I have a C128 with 1541 and 1571 purchased abroad (120V). I have a problem with the little used 1571 which since new has always been extremely noisy, especially with 128 disks. Of late, with 128 autoload disks, the system locks up after it has display "BOOTING ...". Sometimes it also enters the monitor automatically. When I attempt to boot manually the following is displayed "73 CBM DOS V3.0 1571 0" - a DOS mismatch!! The system boots up normally using a 128 CP/M System Disk! Now the problem becomes curiouser and curiouser. Using disks on the 64 mode the system operates normally with the following exceptions. Firstly, a CDU disk cannot be loaded using the standard LOAD"MENU",8,1 and secondly, the directory cannot be obtained in the 64 mode but by employing 128 mode only. CDU disks can be loaded only by using the format LOAD"*",8,1. I have closely read the User's Guide to the Commodore 1571 drive and I feel I must have overlooked something. I have continually checked all the earth connections in the system but I have been frustrated in trying to overcome the problem. As I am an OAP I haven't got much spare cash to have my 1571 checked out by any of the advertisers in the magazine and I feel that there cannot be much wrong with the drive as I rarely use it. Could you possibly help me or, at least, give me some advice? Is there a publication and/or a disk specifically for setting up a 1571? At the present time I will just carry on with the painfully slow but dependable 1541! I remain very puzzled and frustrated.
W.H.Mercer, Co.Durham.

Dear Mr.Mercer,
First of all I think we can rule out alignment problems. If there was something wrong with the disk drive, one would expect it to be consistent in both 64 and 128 mode. The fault is most likely to be the power supply. You say that it is a "foreign" 1571 and uses the 120V supply of another country. I presume that you have the correct adaptor to convert the voltages, but do you have a piece of equipment that takes into account the probable difference in frequency of the mains in this country to that in the country for which the drive was intended? I suggest that first of all you check that both the correct voltage and the correct frequency of supply is getting to the drive. If everything is fine then the only thing that I can suggest is that you try and find the cheapest source of repair I'm afraid. Alternatively you could purchase a Service Manual from FSSL. They cost around twenty pounds for the 1571 but I'm not sure how much one would help you. Perhaps, if you don't really use the drive a lot, you could consider selling it to somebody at a reduced cost, telling them of course that it needs looking at. When operated in 64 mode, the 1571 is exactly the same speed as the 1541 - only in 128 mode does it come into its own. Sorry I can't point the fault at anything more specific.

SMOOTH SCROLLING

Dear CDU,
I have been buying CDU for about eighteen months now and think that it is very worthwhile. I treated the idea of Techno-Info with some scepticism at first after I had seen the advice pages in other magazines. But now I read the section every month and wouldn't be without some of the advice and tips that you have given. Thanks! I'm even writing to it myself now - and it is because I am planning on writing a game for the C64 and would like to include a

smooth scrolling map. The basic idea is that there is a town or other landscape around which the player must travel. By using raster interrupts I can quite easily get the screen to scroll up, down, left and right. But I have two problems. The biggest comes with the vertical scrolling. Because the score and status panel is at the top of the screen, the scrolling starts about five lines down. With the horizontal scrolling it is easy to make the screen 38 columns and thereby cover up the area where the characters appear. And vertically I can shrink the screen to 24 rows but this does not cater for my needs at having the vertical scroll start five lines down. How can I get rid of the flicking? Is there a routine to "confuse" the rasters or something as I have tried for ages and think I must be overlooking something simple. The second problem is that I want to make the game so that the map can be scrolled in more than just four or eight directions. It would be nice to be able to say that it features '360 degree' scrolling. But how exactly is the effect achieved? I would be ever so grateful of any assistance that you could provide.
E.D.Lyons, Liverpool

Dear Mr.Lyons,
I must firstly thank you for your kind comments about CDU and Techno-Info. It really is great to know that the work is appreciated and I hope you continue to find it useful. The secret behind overcoming your first problem is the humble little sprite. If you make one sprite definition into a solid block and expand it horizontally, this can then be duplicated seven times across the screen - end to end - and placed at the correct vertical position. This produces a bar of sprites which will appear above the "flicker" and conceal the characters as they appear and disappear. To make it seem that nothing is there, make the sprites the same colour as the background. As to how to achieve the effect of 360 degree scrolling, it is all a matter of how many pixels are scrolled vertically, each time you scroll one pixel horizontally or vice versa. If you have some sort of vehicle or person that can be 'rotated' and you have, say, sixteen different positions and therefore directions for this thing, then you simply change the number of pixels scrolled horizontally and vertically. For straight up you scroll two up and none to the sides, for right you scroll two to the right and none up or down. For northeast, you scroll two up and two right, and here it comes; for somewhere between the two you would scroll two up and one right, or three up and two right. For just below east you would scroll two right and one down or something. You would proceed to two right and two down for SE, one right and two down for SSE and two down and none to the sides for S, then for SSW you do one left and two down and so on and so forth. With a bit of luck you can get on with writing the game now!

UPDATE

This month's update concerns the CHEQUE BOOK ORGANISER program by PETER WEIGHILL OF BOURNE that appeared recently in CDU. He has very kindly sent a list of lines to change/add to the program that will enhance it for you. One of them cures the problem that a few people experienced where the balance read something like 12.3399999 instead of 12.34. This is actually a rounding bug in the computer itself and NOT in the program. You can see the rounding error by just entering the simple line: PRINT 7.89-7.42 from BASIC. So here goes with the changes:

LOAD"CHEQUE BOOK2", then LIST 11050 and change the "10020" at the end of the line to "11020". Then add the following lines: 11041 IF A$="," OR A$=":" THEN 11020, and 15001 BA=BA*100+.5, and 15002 BA=INT(BA)/100. Then SAVE"@0:CHEQUE BOOK2",8. If you want to enter details for PAY INS like you can for PAY OUTS then enter these lines before saving the program back to disk: 120 REM, 132 PRINT:PRINT"INPUT DESCRIPTION:":GOSUB11000, and 134 CQ$(NO)=B$

Thanks for those changes, Peter. You should find that the program is even better than before now, so make the changes before you forget.

THE TECHNO-INFO CHALLENGE

In March I presented you with a program that could reel off the 2261 prime numbers between 3 and 19997 inclusive in about 355 seconds. The question was this: Can anyone do better? Well I am pleased to say that I had a great response and many people spotted the places where the program could be made to run faster, and in some cases, a lot faster. The overall winner, though, was PAUL GANDER OF GOSPORT who managed to alter the algorithm a bit and generally get the program to run faster. His version calculated all of the prime numbers in less than 228 seconds - cutting the time by over 35%. You will find his program on this issue's disk filed as "PRIMES WINNER". Hopefully Paul will have received his prize of ten blank disks by now. This month's prize may be the same, although I may make it something different. After all, variety is the spice of life! The top five entries were received from the following:

1. with 227.67: Paul Gander of Gosport
2. with 236.83: Neil Barton of Cheshire
3. with 266.20: Henk Schouten of the Netherlands
4. with 278.42: Ian Pollard of Oakham
5. with 290.38: Richard Smedley of Sutton-in-Ashfield

Thanks also to everyone else who participated in the Challenge. Please try again this time - you may be the winner!

THIS MONTH'S CHALLENGE

This time the Techno-Info Challenge doesn't rely so much on your mathematical skills, rather your programming talents. I would like you to design a version of that old favourite, NOUGHTS AND CROSSES (or Tic-Tac-Toe). It must allow the user to compete against the computer in an effort to be the winner, with a row of three crosses in any of the acceptable directions. Of course that would be relatively simple and on this month's disk you will find such a program, filed as "DUMB OXO". It is called "DUMB" because it can't think for itself. The computer's move is entirely random and you may find it difficult to actually get the computer to win! What I want you to do is write from scratch an 'intelligent' Noughts and Crosses game that will be unbeatable. The computer must NEVER lose (well, it must be over 90% successful!). It is fine if the computer draws with the player or wins, but if it loses too much of time you will have to think again about the programming! It's not that difficult and I'm not after straight copies of solutions that have appeared in publications in the past. Oh, by the way - the computer is not allowed to pause when it is "thinking" for more than ten seconds. Do you think you could do it? Have a go anyway and you could steal the limelight. The winner will be declared in the September 1991 issue and the prize will go to the person who can write such a game, and of course - to put you under even more pressure - that winner will be the person who provides the SHORTEST successful program, and by shortest I mean the fewest lines, not the least number of BASIC bytes. Please do have a look at my program on the disk and base yours on that one if you like (the method for checking if someone has won can be vastly improved!), but be sure to make it so that your version doesn't allow the player to win. I will be playing each entry 50 times over the weeks and any where the computer wins (with me trying desperately to beat it of course!) more than 45 times out of those 50 will be eligible for the prize. Then it just comes down to who wrote the shortest version. By the way, no cheating by using extended BASICs or crunching programs that allow extra long lines.

Please send you entries to the normal CDU Techno-Info address (given at the end) but clearly mark your envelope with the word CHALLENGE. Final judging will be done on 30th June 1991. If you can, please send a disk or tape with your entry on. I guarantee that all will be returned within about a week. Best of luck to you all!

THAT'S IT!

Unfortunately that is all we have space for this month, but if you have any programming or hardware queries then please don't hesitate to write to us for assistance. Don't forget to keep sending those tips in either, and be sure to have a go at the Challenge if you feel up to it. Remember, entries by the end of June please. The address for your queries, tips or entries is as follows:

CDU TECHNO-INFO,
11 Cook Close,
Brownsover,
Rugby,
Warwickshire,
CV21 1NG.

See you all again in July.

ADVENTURE HELPLINE

JASON FINCH concludes his help with CDU's ASTRODUS AFFAIR

Twelve months ago CDU started the Adventure Helpline series and today it is still going strong thankfully. Another series sees its first anniversary! Anyway, this month we will conclude the detailed help of THE ASTRODUS AFFAIR. Hopefully you will notice the map of the craft that I have provided here for you this month. It should just give that extra bit of help to show you where you are heading. If you have any further problems relating to this adventure or perhaps you have missed an earlier article which gave details of solving the problem that you are having difficulties with then please write to me at ASTRODUS HELP, 11 Cook Close, Rugby, Warwickshire, CV21 1NG. I'll then do what I can to sort out the problem for you. This months dose of help will guide you to the fuses, thus enabling you to repair the damaged drive and set off into space, having completed the adventure. Let's get on with it then.

ANOTHER DRYGAR

The fuses are contained within a safe in location 23. Upon entering that room you will be devoured by a hungry drygar unless you are carrying a laser gun. The initial problem is that the laser is out of reach in the supplies room but this should have been solved by the fact that the SLOFT grabs it earlier and then discards it on the floor. You can then pick it up from location 21, make your way to cargo bay one, location 23, and fire the laser at the drygar.

THE DETAILS

From where we left off last month, location 4, you should go NORTH, then NORTH again, DOWN the stairs, WEST, SOUTH, DOWN to the bottom level, and then WEST. You should then be able to just GET the LASER. Then go EAST, SOUTH and SOUTH. FIRE the LASER to kill the drygar.

THE SAFE AND FUSES

Now you are free to open the safe by entering the correct combination. If you read GONTRA's log book then you will have noted the entry "COMBS3468279". You are told that this has been scribbled out and so is not completely readable. I will tell you that the combination is actually eight numbers long. Think about the 'S' in "COMBS". When you have opened the safe, retrieve the fuses and make your way to the engine room, location 10. Change the fuse and then return to the middle level and the control room. Press the two buttons that you have previously been unable to press and you are away! Home and dry. If you want to do that little bit on your own then please skip the next section because I'm about to spoil the fun!!

EXACTLY HOW

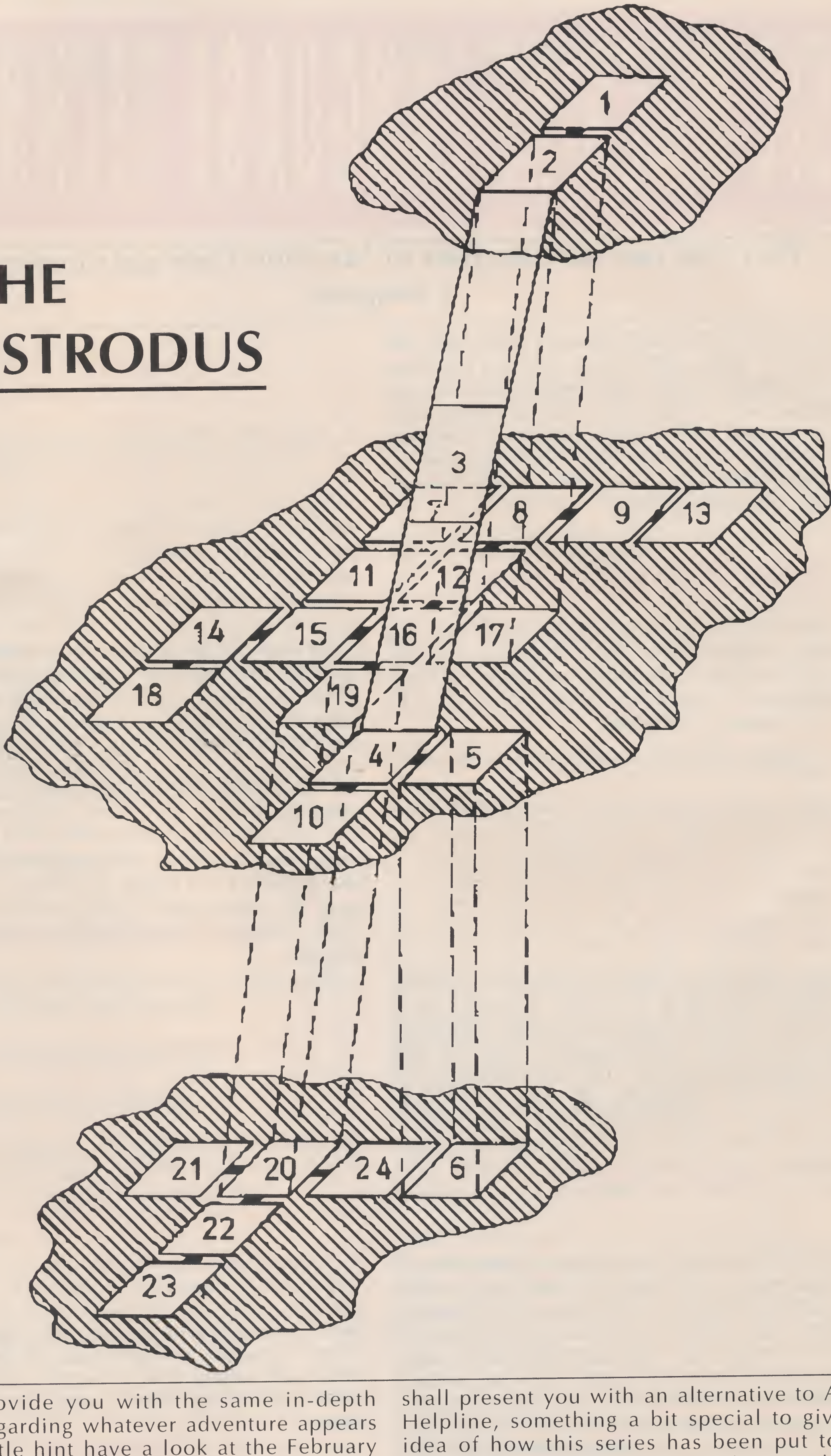
Here's exactly what should be done. The combination is entered by inputting TYPE 53468279. The letter 'S' in the log book was in fact the number '5'. Now you can GET the FUSE. Go NORTH, NORTH, UP, NORTH, EAST, UP, then SOUTH, SOUTH and SOUTH again to reach the engine room. Then CHANGE the FUSE and return to the control room by going NORTH, NORTH, NORTH, DOWN, WEST to cross-section A, WEST again, and WEST once more. Finally, enter the control room to the SOUTH. Now PRESS XX2V and PRESS ZA7Q. The fuse is repaired, the generator is on, the craft is stable and so you are free to zoom off - "The Universe is yours for the taking!"

THAT'S IT

Well there isn't really a lot more to add to the detailed solution - you should now be able to complete the adventure without too much difficulty. Now we have finished with THE ASTRODUS AFFAIR and so in August we will be free to move onto something else. I shall hopefully

[illegible]

THE ASTRODUS



be able to provide you with the same in-depth information regarding whatever adventure appears then - for a little hint have a look at the February 1991 issue! I just hope that I haven't spoilt the surprise of what happens in the end! Next month I shall present you with an alternative to Adventure Helpline, something a bit special to give you an idea of how this series has been put together. I hope that you will appreciate the revealing of my secrets!

BASIC MACHINE LANGUAGE TECHNIQUES

Part 3 of our introduction to Machine Code gets underway J Simpson.

Last Month saw us safely through addition, subtraction and multiplication. I hope, with the use of either the trace program, or using the 'hand' trace method you are now fully conversant with multiplication. This was designed to help you understand such a typical program in complete detail (extremely important when learning how to program) and to help make you familiar with how routines can be constructed and executed.

MORE MULTIPLICATION

By now you should have familiarised yourself with an understanding of how the shifts work. These are a fast and reliable method of both multiplication and division so long as the multiplicand or divisor is in powers of two, (2, 4, 8, 16, etc). Let's examine an example:

Program Segment:

```
100                 LDX #2
110 LOOP
120                 ASL VALUE
130                 DEX
140                 BNE LOOP
150 ; ***  REST OF PROGRAM
```

In this program segment we need to multiply the variable data byte called VALUE by 4. So in Line 100 we set the X index to 2 and in Line 130 and 140 we shall decrement the X and branch back to LOOP until X = zero - in this case two iterations of the loop. The first iteration of the loop will, at Line 130, shift all the bits of the VALUE byte one place to the left, in effect multiplying its value by two. On the next, and final, iteration we repeat the procedure which multiplies by two the new version of VALUE byte once again - thus VALUE has effectively been multiplied by four.

You can see from this that by changing the value of the X Index we can multiply, by repeated shifts, the variable VALUE by either 4,8,16,32,64,128. However, this can lead to a problem because the numerical quantity held in VALUE may overflow after repeated shifting so we must have a method whereby we can easily push the overflow into a further byte. We do this by using the ROL instruction. As the ASL instruction shifts a bit into the carry, then the ROL instruction rotates this bit into another data byte. The following is a short program which will illustrate this:

```
100                 *=$C000
105 ;
110                 LDX #4
120 LOOP
130                 ASL VALUE
140                 ROL VALUE+1
150                 DEX
160                 BNE LOOP
170                 BRK
180 VALUE           BYT 128,0
190                 END
```

If you assemble the program, then enter the monitor and check the two data bytes located at $C00C they should be :80 00 (Low byte = 128 ($80) and High byte = $00). Now if you run the program, (G C000), and again check the two data bytes at $C00C they will now register as :00 08. In other words the High byte is 08 and the low byte is 00. The value being 2048 ($0800 in hex), which is the result of 128 times 16.

Finally when you have multiplication by an odd number, such as three it is a simple procedure to perform without using the complicated multiply routines which, on the whole, do tend to slow processor execution time. For example:

```
TIMES 3
10      LDA POINTER ; LOAD VALUE TO BE
                      MULTIPLIED * 3 INT Ac.
20      ASL A       ; MULTIPLY * 2
30        CLC       ; CLEAR THE CARRY READY FOR
ADDITION
40      ADC POINTER ; ADD THE ORIGINAL VALUE OF
                      POINTER
100     STA POINTER ; STORE THE WHOLE RESULT
                      BACK TO POINTER
TIMES 6 - Insert after 40 above
50      ASL A
```

And so on. Experiment with this idea and you will discover that many multiplications can be performed very rapidly with simple shifts and adds - but then, that is the whole basis of the more complicated multiplication routines.

DIVISION

Using the shifts for division is more or less the same as

multiplication except now we use LSR and ROR to shift the bits in bytes from left to right.
As an example let us divide 300 by 4:

```
100       *=$C000
105 ;
110       LDX #2           ; TWO ITERATIONS = *4
120 LOOP
130       LSR VALUE+1  ; THE HIGH BYTE OF 300 (1*256
                         = $01)
140       ROR VALUE      ; THE LOW BYTE OF 300 (300-
                         256 = 44 ($2C))
150       DEX
160       BNE LOOP
170       BRK
180 VALUE  BYT 44,1     ; (300 - $012C)
```

Run this program and after checking the data bytes at $C00C the result held will be :00 4B or 75 in decimal.

Something more complicated, such as divide 427 by 41 will require more thought to construct a fast and efficient routine, analogous to the multiplication routines of last month. The divisor is successively subtracted from the high order bits of the dividend, and after each subtraction, the result is used rather than the initial dividend and the value of the quotient is incremented by '1' each time. Eventually the result will become negative so we must restore the partial result by adding back the divisor. At this time the quotient must be decremented by '1'. Quotient and dividend will be shifted to the left by one bit position and the algorithm is then repeated.
I'm going to let you experiment with this method and, by comparison with multiplication, come up with a decent routine to perform binary division. Just remember that it is analogous to multiplication.

LOGICAL OPERATIONS

Another class of instructions which the ALU can execute is a set of logical instructions, which include:

AND   OR (ORA) and exclusive OR (EOR)

We shall be discussing these a little further on.

We can include, in addition, the shift operations which we have just been discussing, and the instruction which allows us to make à comparison, namely CMP.

First a look at CMP.
Often we need the program to look at a variable data byte and compare it with a value. As an example we may be utilising memory location $C4 (196) to discover which key the user is pressing at any given time. In Basic we might write a subroutine such as:

```
100 K=PEEK(196)
110 IF K = 64 THEN RETURN : REM NOKEY
120 IF K = 4 THEN F=1     : REM FUNCTION KEY 1
130 IF K = 5 THEN F=3     : REM FUNCTION KEY 3
140 IF K = 41 THE C=0     : REM CURSER UPDOWN KEY
ETC.
```

In Machine Language the same routine could be developed by exploiting the CMP instruction, thus:

```
100       LDA 196    ; CONTENTS OF LOCATION 196
110       CMP #64    ; VALUE IF NO KEY PRESSED
120       BNE SKIP1   ; NOT EQUAL TO 64, THEREFORE
KEY IS PRESSED SO SKIP
130       RTS            ; IS 64 SO RETURN FROM THIS
SUBROUTINE
140 SKIP1
150       CMP #4     ; VALUE OF F1 KEY
160       BNE SKIP2   ; NOT PRESSED SO SKIP TO NEXT
TEST
170       LDA #1       ; WAS PRESSED SO LOAD THE Ac
WITH 1
180       STA FUNC   ; AND STORE AT DATA BYTE FUNC
190       RTS        ; THEN RETURN
200 SKIP2
210       CMP #5     ; VALUE OF F3 KEY

Etc., etc.
```

Another example of comparison:

```
100       LDA #100
110       CMP DATABYTE
120       BEQ NEWPOS
130       RTS
140 NEWPOS
150       REST OF PROGRAM
```

But what is it, exactly, that is taking place here?

When the processor is instructed to make a comparison it will compare the bits of the Ac with the value of the byte it is making the comparison with. For example if we load the accumulator with the content of memory location 196 and the user has not hit a key then the bit pattern will be '01000000'. Our first test was to compare this with the value 64, the bit pattern being '01000000'. because the comparison of bits equalled each other then the ALU will set (1) the 'Z' bit (flag) in the Status Register (SR). If the comparison of bits had not matched then the Z flag would have been cleared (0). The next instruction test the Z flag - BNE (Z flag = 0) or BEQ (Z flag =1), and branches or not accordingly. And so it is we pass through a series of comparisons until we hit jackpot!

SUBROUTINES

A block of instruction which have been given a name by the programmer are the basis for the subroutine concept. Unless the block of instructions is going to be used repeatedly there would be very little point in using a subroutine call and return instruction - this would simply waste valuable processor time. Although I am stating the obvious the advantage of subroutine calls is that the programmer need only write the block of instructions once and then use it repeatedly thus saving memory space and simplifying program design.
The instruction which calls the subroutine is JSR (Jump to Subroutine) and when the program execution reaches the

end of the subroutine it reads the instruction RTS (ReTurn from Subroutine). However, it must know where to return to. It does this by placing (or pushing) the memory location of the next instruction after JSR onto the Stack, and when it reaches RTS, it then withdraws (or pulls) the address from the stack and places this into the program counter. Remember from earlier how I told you that the program counter always keeps tabs on the actual memory address of the program instructions as they are executed. The PC is always pointing to the next instruction to be executed. The processor uses the PC to know where it is at, and so when an RTS is encountered the PC is made to point back to where it was when the JSR was executed.
Most of the programs we have developed and are going to develop during this series would usually be written as subroutines. For example the multiplication routine would be a subroutine which would be called from different parts of the program, or many time from within a main programming loop. It is therefore convenient to define a subroutine whose name would be relevant to its action, in this case MULT. At the end of the routine we would emplace an RTS instruction (Line 210 in the multiplication routine of last month).
Finally, other great advantage of structuring portions of a program into identifiable subroutines is they can be debugged independently, have a mnemonic name, such as MULT, and can be saved independently forming a comprehensive library. Thus, when you come to starting another project you will already have a great bulk of the programming detail complete before you even start! Library routines can consists of such programs as multiplication, division, add and subtract routines, sprite manipulations, joystick manipulations, text printing routines, scrolling effects, Interrupt algorithms, and etc., the list is almost endless.

BACK TO LOGICAL OPERATIONS

The classic logical operations are the AND ORA and EOR. Let's look at each of these and clarify them more fully.

TRUTH TABLES

Each logical operation is characterised by a truth table which expresses the logical value of the result in function of the inputs.

```
AND  0 AND 0 = 0    OR  0 OR 0 = 0    EOR  0 EOR 0 = 0
     0 AND 1 = 0        0 OR 1 = 1         0 EOR 1 = 1
     1 AND 0 = 0        1 OR 0 = 1         1 EOR 0 = 1
     1 AND 1 = 1        1 OR 1 = 1         1 EOR 1 = 0
```

LOGICAL AND

You will notice that the AND operation will return a '1' if both operands are '1'. In fact the operation is characterised by the fact that any other output will return a zero. We use this feature mainly to clear bit positions in a byte, and it is commonly referred to as "masking". For example you might need to mask out the four left-most bits, or high nibble, of a byte, and this could be achieved by ANDing the byte with the bit pattern - '00001111'

```
LDA DATA      ; DATA CONTAINS '10101101'
AND #%00001111 ; the use of #% is used to represent a binary number.
```

The result of this would be to leave the Ac with the value '00001101'.

```
LDA DATA      ; DATA CONTAINS '10101101'
AND #%10000001
```

Result '10000001'

```
LDA DATA      ; DATA CONTAINS '10101101'
AND #%01111110
```

Result '00101100

If you wanted to alter the content of data then the next instruction would simply be STA DATA. One common use of this method, for example, is when Sprites need to be turned off

INCLUSIVE OR

The inclusive OR operation is characterised by the fact that if any one of the operands is a '1', then the result is to set any bit in a byte to '1'. The converse of the AND operation and this is used when we wish to turn bits on.

```
LDA DATA      ; '10101101
ORA #%11110000
```

Result in Ac '11111101' - in other words we have turned on bits '4' and '6'. This operation can be used to turn sprites on.

EXCLUSIVE OR

EOR stands for "Exclusive OR". This differs from the "Inclusive OR", which I have just described, in only one respect - the result is '1' only when one, AND ONLY ONE, of the inputs is equal to '1'. If both inputs are equal to '1' then the result is '0'.
The EOR is used for comparison tests. If any bit is different, the "exclusive OR" of the two bytes will be non-zero. As well as this the EOR is used to compliment a byte since there is no compliment instruction. We do this by performing an EOR of a byte using all bits set. For example:

```
LDA WORD      ; WORD = '10101010'
EOR #%11111111
```

Result in Ac '01010101' - the one's compliment.

AN EXAMINATION OF BRANCHES

So far we have been dealing pretty exclusively with only two conditional branch instructions, namely BEQ and BNE. I've already explained that the result of a compare will either set or clear the 'Z' flag of the status register (SR). Now we shall look at other instructions which can cause a conditional branch.By conditional branch I mean a branch

to another segment of the program conditional upon a certain test being performed and the branch resulting upon the condition of that test. Unconditional branches are JMP and JSR. No test is required and both instruction require a sixteen bit memory address to jump to.
Testing is almost exclusively performed upon certain flags of the Status Register. Let's look at these again:

```
7 6 5 4 3 2 1 0
_______________
N V - B D I Z C
| |   | | | | CARRY
| |   | | | ZERO
| |   | | INTERRUPT
| |   | DECIMAL
| |   BREAK
| OVERFLOW SIGN
(NEGATIVE)
```

Tests are carried out on the Z, C, N, and V flags. We have discussed the Z flag which is set if the result of a compare is equal <BEQ> and clear if the result is not equal <BNE>. We have also seen, when we used the shift instructions, the C flag in use to detect the state of a bit shifted into the Carry <BCC> or <BCS>. We also use the Carry flag to detect the result of an addition or subtraction in that should the value of a byte increase or decrease beyond the byte parameters (0-255) then the C_Flag is set (addition) or cleared (subtraction). Again we use this information to conduct a branch, as in the following examples.

```
EG.1  LDA MEMADDRESS   ; PLACE THE CONTENT OF
                         MEMADDRESS INTO Ac
      CLC   ; CLEAR THE CARRY BIT READY FOR
                ADDITION
      ADC #$28  ; ADD THE VALUE OF 40 TO THE Ac
      STA MEMADDRESS   ; PLACE IT BACK INTO
                         MEMADDRESS
      BCC SKIP  ; CHECK THE CARRY FOR AN
                  OVERFLOW, IF NO JUMP TO SKIP
      INC  MEMADDRESS+1 ; IF IT IS ADD 1 TO HIGH
                  BYTE OF MEMADDRESS
SKIP
    Rest of Program...

EG.2  LDA MEMADDRESS   ; PLACE THE CONTENT OF
                         MEMADDRESS INTO Ac
      SEC           ; SET THE CARRY BIT READY FOR
                      SUBTRACTION
      SBC #$28      ; SUBTRACT 40 FROM THE Ac
      BCS SKIP      ; CHECK CARRY FOR UNDERFLOW,
                      IF NO JUMP TO SKIP
      DEC  MEMADDRESS+1 ; IF IT IS SUBTRACT 1
                      FROMHIGH BYTE OF MEMADDRESS
                      SKIP
    Rest OF Program
```

The 'N' flag is set whenever the result is negative in two's compliment (this is signed arithmetic - see article 'Numbers and Bytes', CDU Oct.90/Jan.91). In practise the 'N' flag is identical to bit '7' of a result. It is set or cleared by all data transfers and processing instructions, This means we can very conveniently, and rapidly test bit 7 of the Ac, X and Y registers by referring to this flag. The two branch instructions which use this test are BPL (Branch if result PLus), and BMI (Branch if result MInus). We can exploit this, for example, during loop sequences of our programs.

```
    LDX #39
LOOP
    LDA TMP
    CLC
    ADC #2
    STA TMP
    DEX
    BPL LOOP
```

Here we will obtain 40 iterations of the loop. Each iteration of the loop will decrement the X register until it reaches zero where on the next iteration it will flip the register to 255, thus setting the 'N' flag, and making the truth in the branch (BPL) false and executing no more branches. Because the N_flag is set/cleared dependent upon the 7th bit, it follows, when using this bit to determine if a countdown has dropped below zero, that the start value of the count must be lower than 128 ($80). This flag also gives us a useful 3-way switch. A byte can be set to either 0, 1, or 255. An example of this in use is to use a byte to reflect the state of a joystick direction. DY = Y direction of joystick. If DY = 0 then no action if DY = 255 then LEFT else RIGHT.

```
    LDA DY
    BEQ RETURN
    BMI LEFT
RIGHT
    ...
    ...
    RTS
LEFT
    ...
    ...
RETURN
    RTS
```

Finally the 'V' flag, or overflow. When we are dealing with signed arithmetic bit 7 is used to indicate either a positive (bit clear), or negative (bit set), number. This gives us a numerical range within a byte of -127 to +128. When we perform math on bytes, we need to detect when the value overflows the range, much the same as we use the 'C' flag. Now, however, we use bit 6 to indicate that the result of the addition or subtraction of two's compliment numbers might be incorrect because of an overflow from bit 6 to bit 7, i.e., the sign bit. This technique is fully explained in the before mentioned articles, Numbers and Bytes. During the normal course of programming it is a branch that is not often used.

ADDRESSING TECHNIQUES

It is now time to switch our attention to a general theory of addressing techniques which have been developed in order to retrieve data.

Because the 6510 has no 16-bit registers, apart from the program counter, it is necessary for us, as programmers, to understand the various addressing modes available, and in particular the use of the X and Y index registers. At first sight some of the retrieval methods might seem complex - but, like all things - it can only get easier with practise!

1. IMPLIED ADDRESSING - This is where a single byte instruction, such as TAX, operates upon an internal register. It doesn't require the address of the operand on which it operates. Rather, its opcode specifies one or more internal registers. as an example, TAX transfers the contents of the accumulator into the X index register.
Instructions which operate exclusively inside the 6510 are: CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, NOP, SEC, SED, SEI, TAX, TAY, TSX, TXA, TYA.

and - BRK, PHA, PHP, PLA, PLP, RTI, RTS are instructions which require memory access.

2. IMMEDIATE ADDRESSING - As you now know, the 6510 has only 8-bit working registers (the program counter (PC) is not a working register), and so all immediate addressing is limited to 8-bit constants. This means that all instruction are two bytes in length. The first byte is the opcode, and the second byte will contain the data. An example of this could be ADC #25 - The Opcode and first byte is ADC and the second byte contains the literal value of 25 to be added to the Ac. LDA #255 First byte and Opcode is LDA and the second byte is the numerical value of 255 to be contained in the Ac.
Instruction using this mode are: ADC, AND, CMP, CPX, CPY, EOR, LDA, LDX, LDY, ORA, SBC.

3. ABSOLUTE ADDRESSING - Here three bytes are required. The first byte is the opcode, and the next two bytes make up the 16-bit address specifying where the operand is located. LDA $2000 specifically tells the processor to copy the data contained in address $2000 and deposit it into the Ac. By now I'm sure you are aware that the two bytes following the opcode are in High/low byte order - this means that the three bytes, shown in hex, occupy memory thus:

```
$C000  A9  <instruction mnemonic - opcode>
$C001  00  <low byte of 16-bit address>
$C002  20  <high byte of 16-bit address>
```

Absolute Instructions are: ADC, AND, ASL, BIT, CMP, CPX, CPY, DEC, EOR, INC, JUMP, JSR, LDA, LDX, LDY, LSR, ORA, ROL, ROR, SBC, STA, STX, STY.

4. ZERO-PAGE ADDRESSING - In this mode, which is in essence the same as Absolute, only two bytes are required. The first, as usual, is for the opcode and the second is an eight-bit address. The first 256 memory locations, as you can see, are contained within a single byte (00-FF). Now this offers us with very significant speed advantages as well as shorter code and it should be used wherever possible. However, this does require very careful memory management by the programmer. Viewed generally, the first 256 locations can be seen as a set of working registers for the 6510, and any instruction will execute on these 256 'registers' in just three clock cycles. This space, because of its limited size, should therefore be reserved for essential data which requires high-speed retrieval. The system itself makes extensive use of zero-page, especially for the Basic Interpreter. We will come back to the zero-page concept shortly.
Zero-page instructions are the same as absolute with the exception of JMP and JSR.

5. RELATIVE ADDRESSING - This method uses two bytes, the first of these is a form of jump instruction and the second specifies the displacement and sign. To differentiate from the normal jump instruction, these are labeled as 'branches', and always use the Relative Addressing mode (plus other sub-modes such as Indexed and Indirect - discussed shortly). The timing of these instructions is flexible depending on certain criteria - for example, when a test fails and no branch occurs then only two cycles take place but if the test is successful then three cycles will occur, however, if the branch crosses a page boundary then a further cycle is added bringing the total to four. Usually we are not too concerned about this but if the branch is a part of an exact or critical timing loop then caution must be exercised.
The displacement of the branch is +128 bytes forward from the instruction or -127 bytes backwards. Let's examine a couple of examples to show this more fully:

```
EG.1  ...  BYTE COUNT  OF DISPLACEMENT
      BEQ SKIP ($05)
      LDA #1          00, 01
      STA $4000       02, 03, 04
SKIP

EG.2  ...
LOOP
      ADC #40          FB
      STA $1234,X      FC, FD, FE
      INX              FF
      BNE LOOP ($FA)
```

Instructions which implement Relative Addressing are the branch instructions which test flags within the Status Register (SR): BCC, BCS, BEQ, BMI, BNE, BPL, BVC, BVS.

6. INDEXED ADDRESSING - As you know the 6510 is equipped with two index registers, the X and the Y. However, these registers are limited to 8-bits thus the 6510 only supplies us with a limited capability of indexed addressing. What actually takes place is that the index is added to the address field of an instruction: STA $1234,X which means STA $1234 + whatever the value of X might be.

```
      LDX #2
LOOP
      TXA
      STA 1234,X
      DEX
      BPL LOOP
      ...
Result at end of loop  Location $1234 = $00
                       Location $1235 = $01
                       Location $1236 = $02
                             Ac = $00
                          X Index = $FF
```

Frequently one or the other of the index registers is used as

a counter in order to access elements from a table of data. Usually most tables are less than 256 bytes long and so problems do not occur. If the table is longer than 256 bytes then an alternative indexing algorithm must be used which will increment the high byte of the table address once the Index flips back to zero.
The indexed addressing mode can also be used with the zero-page addressing mode, i.e., with an 8-bit address field.
The list of instructions which may use this mode are: (With X index) ADC, AND, ASL, CMP, DEC, EOR, INC, LDA, LDY, LSR, ORA, ROL, ROR, SBC, STA. (With Y index) ADC, AND, CMP, EOR, LDA, LDX, ORA, SBC, STA.

7. ABSOLUTE INDIRECT - There is only one instruction in this mode, namely JMP. An indirect jump transfers program flow control to a new address. It isn't used that much by programmers due to a 'bug' in the '6502' series chips. If the indirect jump address is located on a page boundary - for example $40FF - program flow will be transferred to an erroneous address. How does it work? Suppose that the first two bytes of zero page contain the value 34 80 and you used the instruction JMP ($0000) this would have the same affect as using JMP $8034. This instruction is useful when a table of addresses (such as the three vectors at the top of memory) exist in a block. For example the reset vector at $FFFC and $FFFD can be called by JMP ($FFFC). Or, if you use your monitor to look at the ROM load routine situated from $F49E you will see at $F4A2 the instruction JMP ($0330), now by checking the vector address at $0330 and $0331 you will find the byte values of $A5 and $F4, or address $F4A5 which is the next address after the indirect JMP instruction. Can you figure out why the program is constructed like this?
8. INDIRECT INDEXED - In this mode the content of the Y Index register is added to the zero page address to retrieve the final 16-bit address. An example of this is LDA ($00),Y. This means that the accumulator is loaded from the address of the consecutive zero-page bytes offset by the value of the Y index. Let's look at a more detailed example. We really need to understand this concept thoroughly in order to create successful programs to shift large quantities of data.

```
90  CLEAR
95 ;
100    LDY #0
110    LDA #4
120    STY $FB
130    STA $FC
140 LOOP
150    LDA #32
160    STA ($FB),Y
170    INY
180    BNE LOOP
190    INC ($FC)
200    LDA $FC
210    CMP #8
220    BNE LOOP
230    RTS
```

In lines 100 to 130 we have loaded two zero-page address, namely $FB and $FC with the values of 0 and 4 respectively. What this means is that we have loaded the screen's base address $0400 into high/low order zero-page bytes. Using the monitor to view locations $FB and $FC we would see:

```
M:; 00FB  00 04 00 00 00 00 00 00
```

Line 140 labels our branch as LOOP.
Line 150 now loads the accumulator with the value of 32 which is the ASCII representation of a space.
Line 160 stores the value held by the accumulator into the 16-bit address in $FB/FC (00 40) which is $0400 offset by the value of Y. We know that at the start of the loop Y was equal to zero from line 100. So we store 32, or a space, at the top left corner of the screen $0400 + 0 (Y)
Line 170 and 180 increments the Y register and tests to find out if it has counted through 256 iterations and arrived back at zero - if not then we branch back to loop and deposit another space on the screen ($0400 + Y). If Y has flipped back to zero then the program drops through to line 190.
Line 190 increments the zero-page location, and the high byte of our screen base address, $FC. If we didn't do this then when the program branches back to loop it would proceed to deposit a space back at $0400 which we don't want, so by incrementing the high byte it will now deposit spaces from $0500 + Y. Pretty obvious, eh!
Line 200 loads the Ac with the value currently held in $FC.
Line 210 compares this with 8. We must stop the loop once the screen is full of space characters otherwise it would continue filling memory locations in our programming area ($0800 onwards) with 32s - this could be disastrous!
If you want you can change the space character at line 150 for any other to fill the screen with.
This is not the best solution for filling the screen with any particular character because with this routine we filled 1Kbyte of RAM with a character. The screen area is only 1000 bytes long, and 1Kbyte is 1024 bytes. Those extra 24 bytes just happen to be where the sprite image pointers are located. So, if we are using sprites in our program then we must take this into account when using a screenfill routine.
However, I hope that this example demonstrates the use of Indirect Indexed Addressing. Permissible instructions are: ADC, AND, CMP, EOR, LDA, ORA, SBC, STA.

9. INDEXED INDIRECT - LDA ($00,X) This mode adds the content of the X index to the zero page address to give a final 16-bit address. Suppose the first four bytes in zero-page are 37 64 8B 42 and that X index contains the value of '2'. Then LDA ($00,X) would be the same as LDA $428B. This instruction is very useful when zero-page contains a table of pointers. We shall deal with this at a later date.
Instructions used with Indexed Indirect are: ADC, AND, CMP, EOR, LDA, ORA, SBC, STA.

FINALE FOR THIS MONTH

Well, that's it for this month - next time we will move deeper into the programming pond when we start to create actual and worthwhile routines which can be saved into a subroutine library.

FRANTIC

As they say in the Royal Navy, if it moves - Blast It, if it doesn't - Salute It

ROY FIELDING

FRANTIC is one of those good old fashioned Blast everything that moves type of games. You are in control of your trusty space fighter and you must battle your way through wave after wave of enemy forces.

The game consists of four levels, and each level has two stages. The first stage is where battle against the enemy fighters takes place, and stage two brings you in confrontation with the Alien.

Plug your joystick into Port 2 and let battle commence.

COMPUTER INTELLIGENCE

Aspects of Computer Intelligence explored STEVEN BURGESS

A computer in its raw form cannot really do a great deal in the nature of intelligence. In actual fact it can do nothing. It is vastly stupid, in this respect.
In order to implement a form of artificial intelligence on a computer we must first establish how real intelligence works. How WE come to conclusions.
This article and its program concentrates only on the factual type of intelligence and not on compromisial decision making. Thus anything that the computer can tell you is based entirely on what it knows. If it not sure - it does not have enough information or none whatsoever - an answer will not be given, it will simply report I DON'T KNOW.

HOW DO WE DO IT

When we have to make a decision or answer a question, in the ideal world we have sufficient information to come to or answer it. We think, say, of CAT and immediately MAMMAL and FOUR LEGS enter our head. So if the question was DOES A CAT HAVE FOUR LEGS? we could answer YES. But if the question DOES A CAT HAVE TWO LEGS? were posed, what do we do then? Our mind has in it MAMMAL and FOUR LEGS and we scan through that and do not come across TWO legs. Why do we not answer I DON'T KNOW?

MUTUAL EXCLUSIVITY

We all know that we would answer NO to this question and may even further it by adding A CAT HAS FOUR LEGS. This is because A CAT HAS FOUR LEGS and a CAT HAS TWO LEGS are mutually exclusive responses. A cat cannot have only two legs and only four legs at the same time. It must have one or the other. So if our memory tells us that a cat has FOUR LEGS we can assume that a cat does not have TWO LEGS. If of course the question was a trick question then we would have to think again. A cat which has four legs most definitely does have two legs. It has two sets of two legs, in fact. But we, as humans with many sensory perceptions, can usually tell if the question is going to be a trick question - and if we can't we get the answer wrong unless we are unbearably pedantic when we will be looking for these potential tricks in questions which may not be intended as tricky.

Even by that simplistic answer we can see that the concept of intelligence, even on a fundamental level, can be a very complex one indeed. Not so much for us, but for a computer and a computer programming it can seem almost an unscalable mountain.

ON A COMPUTER

To implement artificial intelligence on a computer we must have an intelligence base. We must have a memory which has information stored in it. Let us take the animal facts example further.

We could have a two dimensional array storing the subject and attributes which that subject has. So we could store:

A$(0,0)="CAT":a$(0,1)="MAMMAL":a$(0,2)="FOUR LEGS"

and printing out the contents of record 0 would display all the information on cats:

```
a CAT is a
  MAMMAL which has
  FOUR LEGS
```

So when the user asked DOES A CAT HAVE FOUR LEGS the computer would scan through the first field of each record until it matched CAT then it would scan through the other fields until it matched FOUR LEGS and if it matched it, which it would, it would reply YES. But, if we asked DOES A CAT HAVE TWO LEGS the computer would match for CAT then try to match for TWO LEGS but it would never be able to match so it would reply I DON'T KNOW. It could not reply NO because it doesn't know the nature of the data stored and cannot tell if TWO LEGS and FOUR LEGS are mutually exclusive. Unlike us, the computer cannot tell from how the question is asked if it is to be a trick question and even if it could the data isn't quantified so it doesn't know that TWO is less than FOUR so it could not answer the trick question. It would claim ignorance.

We could field typify each field so that there was a specific field for number of legs, a specific field for animal type etc so the record would be like this:

RECORD 0

```
ANIMAL : CAT
TYPE   : MAMMAL
LEGS   : FOUR
```

So when the question DOES A CAT HAVE TWO LEGS the computer could answer NO because it now knows that the animal has FOUR LEGS, because it is aware of the nature of the data.
This causes problems when we wish to enter a bird into the knowledge base:

RECORD 1

```
ANIMAL : BUDGIE
TYPE   : BIRD
LEGS   : TWO
```

If asked DOES A BUDGIE FLY the computer would be incapable of telling us because it has no data on ability to fly. It would say I DON'T KNOW. You may say that the computer could take the fact that the animal is a bird and deduce from this that the animal could fly. In this case the computer would reply YES. Let us enter another bird:

RECORD 2

```
ANIMAL : EMU
TYPE   : BIRD
LEGS   : TWO
```

CAN AN EMU FLY? Well of course it can, would herald the computer, it's a bird, isn't it? I think you see the problem.
Here it would be necessary to add another field to the record : CAN FLY? This is okay, but when we begin to add animals with more peculiarities it is going to be necessary to add more and more fields to make their entry more comprehensive and questions more answerable. If not the computer is going to be making generalisations based on other entries - "Of course a dolphin is a pet - it's a mammal isn't it?" and so on.

THE ANSWER MY FRIEND

...is blowing in the trees. The answer is blowing in the trees.
I don't propose delving to deeply into the structure of trees, there have been articles written already to do that, lets just say that trees provide the ideal data structure for our non-compromisial intelligence base. From now on in I will call it a deterministic intelligence base - a) because it is easier and b) I don't actually think non-compromisial is a word.

Trees operate on a top down structure and branch out rather like... well rather like trees, really, except upside down. You start at the top where there is a question. If the answer is YES you branch to the left, if it is NO you branch to the right. This method of data storage is much more desirable because data gets sorted automatically into groups. The problem is the data cannot be typified so the computer will not know when terms are mutually exclusive. Consider this diagram.

```
                    ###########
                    # MAMMAL? #
                  Y ########### N
                #                 #
   #########                          #########
   #  CAT  #                          #BUDGIE #
   #########                          #########
```

If the animal is a mammal then it will be stored on the left hand side, if it isn't it will be stored on the right hand side. You can split these up further and further so you could have a great many different branches - one for mammals with four legs, one for birds, reptiles and so on. You could go on forever and ever. The article CONCEPTS OF DATA STORAGE IN BASIC PROGRAMMING also written by me had a fun program which stored animals in a tree. You have to think of an animal and the computer will ask you questions about it and then guess what animal you were thinking about. This is deterministic reasoning.

The problem with that program was that you could not interrogate the computer, it had to interrogate you. A much more versatile and less tiresome method of finding things out would be if you could interrogate the computer with questions like IS A CAT A MAMMAL. This would have to be made into a more easily understandable sentence for the computer - unless we programmed a parser, but a syntax such as CAT.MAMMAL? to which the computer would reply YES or TRUE, would probably be acceptable. But what would we have to do to be able to do this?

The animal facts program - which is reproduced on this months disk for your perusal - is a top down structure. You can only go from the top down. So you can only go from question to answer. For the above to work we need to be able to go from answer to question. So if the answer is CAT the computer should be able to tell us that a CAT is a MAMMAL by back-tracking through the tree.

But binary trees (see article mentioned earlier) by definition are top down. To combat this problem we have to use a different type of tree. We have to use a tree which has pointers which point to the element before. When this has been done we simply have to search through the entire knowledge base sequentially and then backtrack checking appropriate data and making decisions dependent upon the question asked. Consider this simplistic tree.

```
     1 MAMMAL?
     Y *     * N
       *     *
       *      *
  2 FOURLEGS?  3 BIRD
  Y *   * N
    *    *
   *      *
 4 CAT 5 DOLPHIN
```

So if we asked the question IS A DOLPHIN A MAMMAL? The computer would search through the entire array until it found DOLPHIN then it would backtrack to the element from which dolphin came. It would see that the No pointer takes it back to dolphin so it would know that a dolphin doesn't have four legs. It would backtrack again and see that the LEG question came from the YES pointer so it could then answer YES to the question. YES a dolphin is a MAMMAL. If you asked it DOES A DOLPHIN HAVE FOUR LEGS? It would answer NO because it would be able to see that DOLPHIN came from the NO pointer of the FOUR LEGS? question. The same is true of IS A BIRD A MAMMAL? The computer would find bird in its list and backtrack to the question before. It would see that bird came from the NO pointer and answer NO - a BIRD is not a MAMMAL.

The pointer system for a binary tree involved the use of two pointers - one for YES one for NO. The pointer system for this traversable tree needs three pointers. One for YES, one for NO and one to indicate where the node is connected to. So the stored list for the above tree would be:

n	NAME	YESp	NOp	FROMp
1	MAMMAL	2	3	-1
2	FOURLEGS	4	5	1
3	BIRD	-1	-1	1
4	CAT	-1	-1	2
5	DOLPHIN	-1	-1	2

When YES and NO are both -1 it means it is a terminal node - there is nothing else after it. When FROM is -1 it means that the beginning of the table has been reached.

Whenever I am programming knowledge bases I am usually at a loss as to what to store in them. The animal examples are fine and are great fun for children, but they don't really serve any purpose. You could store information about famous people in them, I suppose, but this still would provide only for fun use. One place where these deterministic systems are often used is in the car maintenance field and in chemistry, but I am clueless about both of these subjects. So, for the purposes of this article and its program I decided to do it on famous and historical people. The database is not really huge, but it serves as an example. You can ask questions like was Hitler Mad? and is Forsyth funny? For some reason beyond my comprehension the computer has a very evil stance and tends to make fun of everybody. I can't think why.

HOW TO USE IT.

When the program has loaded, it takes less than half a minute, you can load in the demo file by selecting option 4 and then typing CELEBS as the filename.

```
1...ADD TO INTELLIGENCE BASE
2...INTERROGATE
3...DETAILS
4...LOAD
```

5...SAVE
6...END

This program is an extension to the animal facts program as it allows you to interrogate the base with questions. It does not test for mutual exclusivity and it is essential to get the spelling of all words absolutely correct and add a question mark to questions.

Option one should only be used to create databases of your own invention. You cannot add to existing complete bases as the pointers are not changed and any base which you add will be standalone and inaccessible.

When you choose option one you will be asked the following questions:

ENTER CONTENTS?

ENTER TRUTH POINTER?

ENTER FALSE POINTER?

ENTER BACKTRACK?

for contents you must enter either a question or an answer. If it is an answer you must enter -1 and -1 for the T & F pointers. For the backtrack you must, in all cases, enter the element which accesses this element. If you are on the first element enter -1.

You can probably see from this that it is necessary to design your knowledge base first. You will need a large piece of plain paper and a pencil in order to do it. You simply design it in a tree like structure and number each element sequentially.

If you enter a question you must enter the elements to which control will go if the answer is TRUE and if the answer is FALSE - these are entered in the TRUTH and FALSE pointers respectively.

INTERROGATE

To interrogate the knowledge base you simply have to type in a question which the computer will try to answer. The question must be simplified and must be composed of SUBJECT.QUESTION?

So to ask IS FORSYTH FUNNY you would type:

FORSYTH.FUNNY?

to which the computer would reply NO. It isn't necessary for questions/answers to be one word providing they question and answer are separated by a . It certainly makes it easier if they are kept as simple as possible, however.
If the computer has insufficient information to answer you it will reply I DON'T KNOW.

DETAILS

This option allows you to see what a person is. So typing FORSYTH would list:

FORSYTH IS

TOUPEE?
NOT FUNNY?
NOT MAD?
MAN?

you should ignore the question marks. The above means Forsyth is not funny, has a toupee, is not mad and is a man.

Well that is it, I'm afraid. I hope you find the programme fun to use and you may even find a use for it. Goodbye.

DIAGRAM 1.0

here is a list of people contained in the knowledge base:

HITLER (Adolph)
KINNOCK (Neil)
HOLNESS (Bob)
FORSYTH (Bruce)
CHARLES (Prince)
MAJOR (John)
SHARPE (Tom)
ARCHER (Jeffrey)
WOODHOUSE (Pelham Grenville)
FRY (Stephen)
LAURIE (Hugh)
WOOD (Victoria)
THATCHER (Margaret)
CURRIE (Edwina)
FRENCH (Dawn)
SAUNDERS (Jennifer)
QUEEN (The)

and the questions

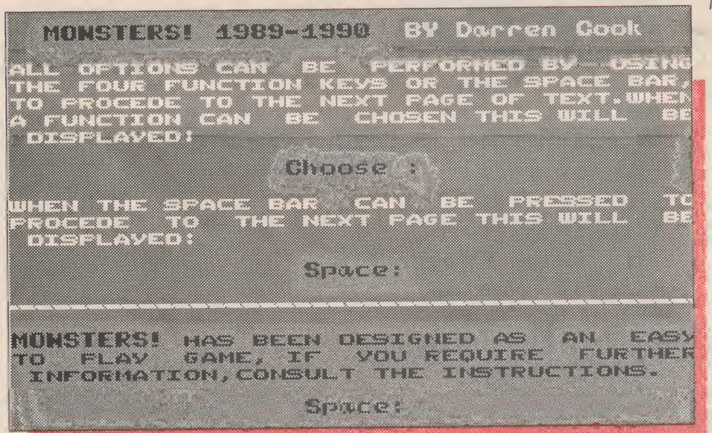
MAN?
MAD?
WELSH?
FUNNY?
TOUPEE?
NOVELIST?
RUDE?
ROYAL?
TALL?
PRIME MINISTER?
WOMAN?
IN GOVERNMENT?
WAS PRIME MINISTER?
FAT?
SINGER?

MONSTERS!

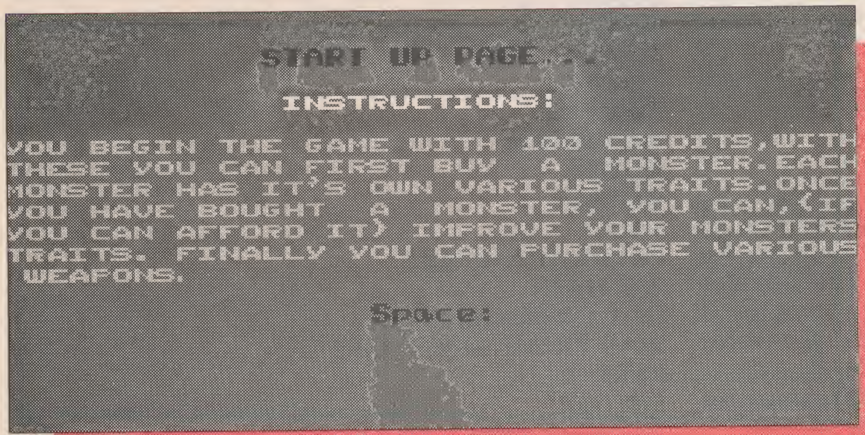
Buying Stocks and Shares was never like this

DARREN COOK

Monsters! is an easy to use fantasy strategy game in which you can buy, sell and improve MONSTERS to fight with other MONSTERS for cash prizes. The aim of the game is to take control of the TWELFTH MONSTER, usually named CYBILL, and earn over 2000 credits so that you can retire.



```
 MONSTERS! 1989-1990  BY Darren Cook
ALL OPTIONS CAN  BE  PERFORMED BY  USING
THE FOUR FUNCTION KEYS OR THE SPACE BAR,
TO PROCEDE TO THE NEXT PAGE OF TEXT.WHEN
A FUNCTION CAN  BE  CHOSEN THIS WILL  BE
 DISPLAYED!

             Choose :

WHEN THE SPACE BAR  CAN  BE  PRESSED  TO
PROCEDE  TO  THE NEXT PAGE THIS WILL  BE
 DISPLAYED:

              Space:

MONSTERS! HAS BEEN DESIGNED AS  AN  EASY
TO  PLAY  GAME, IF  YOU REQUIRE  FURTHER
 INFORMATION,CONSULT THE INSTRUCTIONS.

              Space:
```



```
          START UP PAGE...

           INSTRUCTIONS:

YOU BEGIN THE GAME WITH 100 CREDITS,WITH
THESE YOU CAN FIRST BUY  A  MONSTER.EACH
MONSTER HAS IT'S OWN VARIOUS TRAITS.ONCE
YOU HAVE BOUGHT  A  MONSTER, YOU CAN,(IF
YOU CAN AFFORD IT) IMPROVE YOUR MONSTERS
TRAITS. FINALLY YOU CAN PURCHASE VARIOUS
 WEAPONS.

              Space:
```

FKEYS MAKE IT EASY

The game has been set out in such a way that it should be easy to grasp for beginners and experts alike. All options can be performed using the FUNCTION keys or the SPACE BAR. When a function key must be chosen, "CHOOSE:" will be displayed. When the SPACE BAR should be depressed to proceed onto the next page of text "SPACE:" will be displayed.
You begin the game with 100 credits. The PURCHASE MONSTER menu will be displayed. This is because you need a MONSTER before you can use any of the other options. All options can be selected from the MAIN MENU.

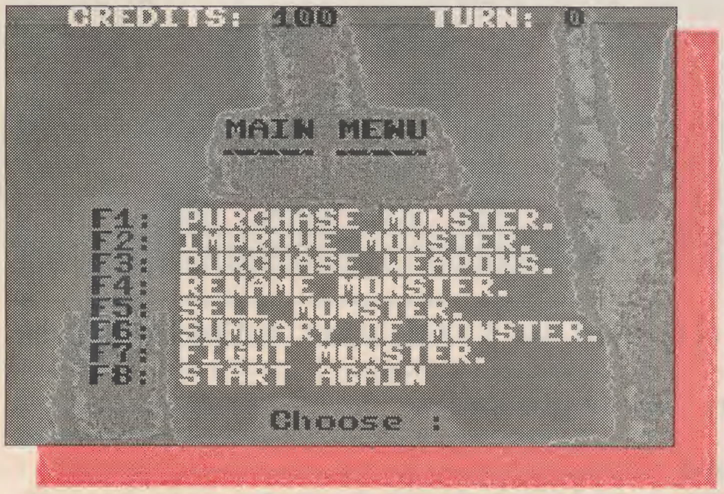
OPTIONS AVAILABLE

F1 - PURCHASE MONSTER

There are 12 different MONSTERs each with their own different traits or characteristics. The higher the traits of a MONSTER, the better fighter the MONSTER will be. You can purchase any MONSTER which you can afford but you must have at least 1 credit to play the game. Therefore, you cannot purchase JOYTRICK on your first turn.

F2 - IMPROVE MONSTER

You can increase any of your MONSTER's traits for 10 credits. The value of your MONSTER increases by 5 for every additional trait. You can reset your MONSTER's traits to what they were when you entered the IMPROVE MONSTER menu. You are refunded the difference of the present cost of the MONSTER and the original cost of the MONSTER.



```
CREDITS: 100        TURN: 0

        MAIN MENU

F1: PURCHASE MONSTER.
F2: IMPROVE MONSTER.
F3: PURCHASE WEAPONS.
F4: RENAME MONSTER.
F5: SELL MONSTER.
F6: SUMMARY OF MONSTER.
F7: FIGHT MONSTER.
F8: START AGAIN

        Choose :
```

F3 - PURCHASE WEAPONS

To increase your MONSTERs fighting potential, you are entitled to purchase different weapons. You are allowed up to eight weapons, but only one can be used in a fight. For more powerful weapons your MONSTER needs to have certain traits to use the weapon properly. Without these traits or the correct cash you cannot purchase the weapon. The weapons are categorised into four different types. PHYSICAL CONTACT WEAPONS, PROJECTILE WEAPONS, PHYSICAL IMPROVEMENT WEAPONS and MIND WEAPONS. Certain types of weapons work better on different MONSTERs. Please note; K.Duster is short for Knuckle-Duster, G's Milk is short for Giraffe's Milk and B.B Food is short for Body-Building Food.

F4 - RENAME MONSTER

You may, if you wish to, alter the original name of your MONSTER. This name will stay the same throughout the game, even if you sell your MONSTER.

F5 - SELL MONSTER

If you want to change your MONSTER for some reason, then you must first sell your MONSTER. Your MONSTER will be sold for it's present value and any weapons which you might also own will also be sold for a percentage of their original value. The traits of your sold MONSTER will remain the same once they are sold but it's value will go up, or down, like in all buy and sell situations.

F6 - SUMMARY OF MONSTER

A summary of your MONSTER will be displayed, including; current value of MONSTER, trait levels, battles won and weapons owned.

F7 - FIGHT MONSTER

Using this option, your MONSTER will be able to fight other MONSTERs, there are four stages;

1) Choose the weapon you wish your MONSTER to use in the fight. If you do not own any weapons, then your MONSTER will just use it's body.

2) Choose an opponent. If you fight the MONSTERs which have higher traits, then if you win, you will earn more money.

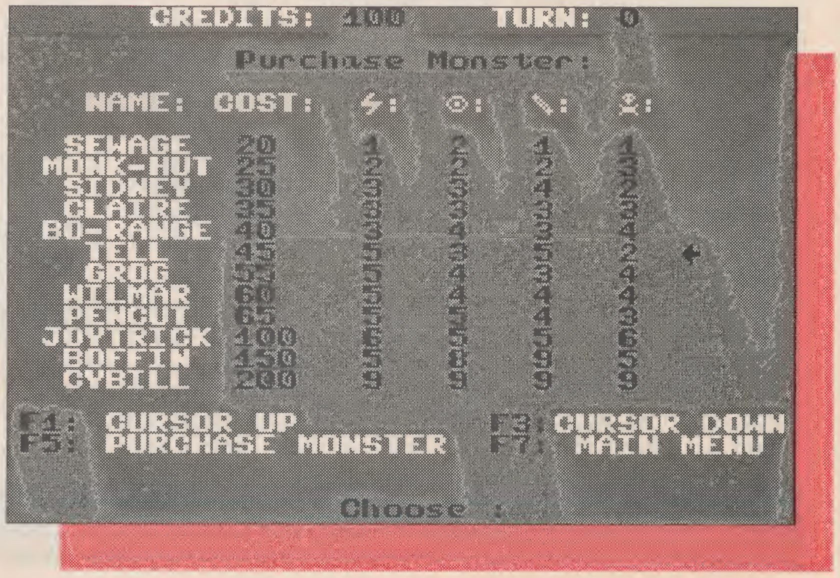
3) You are then offered the chance for your MONSTER to be re-vitalised. When you purchase a MONSTER it is only at 90% health.

4) Finally, you are offered a defence for your MONSTER. This significantly reduces the risk of loosing but they are rather expensive. The stronger the opponent compared to your MONSTER, the more expensive the defence is.

When the fighting sequence commences, you will see your MONSTER on the left part of the screen and it's opponent on the right. The health values of both MONSTERs are displayed at the top of the screen. As each MONSTERs' health decreases so will it. If your MONSTER wins the fight then it's health will be re-vitalised and you will receive a cash prize. However, if your MONSTER loses it will be taken away from you by the doctors, it will be resurrected and put on the market for a higher price, leaving you with no MONSTER.

F8 - START AGAIN

If you are doing rather badly then you might wish to start again. Your will be queeried before starting again.



```
      CREDITS: 100        TURN: 0
          Purchase Monster:
   NAME:  COST:   ⚡:    ◎:    ✎:    ☠:

   SEWAGE   20     1     2     1     1
 MONK-HUT   25     2     2     2     3
   SIDNEY   30     3     3     4     2
   CLAIRE   35     3     3     3     3
 BO-RANGE   40     3     4     3     4
     TELL   45     5     3     5     2    ◄
     GROG   55     5     4     3     4
   WILMAR   60     5     4     4     4
   PENCUT   65     5     5     4     3
 JOYTRICK  100     6     5     5     6
   BOFFIN  150     5     8     9     5
   CYBILL  200     9     9     9     9

F1: CURSOR UP             F3: CURSOR DOWN
F5: PURCHASE MONSTER      F7: MAIN MENU

              Choose :
```

PROGRAM PLANNING

We look at DIY PROGRAMMING and in particular a DATABASE

Steven Burgess

It's very nice, once a month, to be able to buy CDU and have access to top quality programs without having to type them in or write them ourselves. Nobody could deny it. It's the most beautiful concept in the world. But occasionally it is nice to write ones own program. To unleash the creativity inside each and every one of us.

But when we get down to it we sit at our desks, look at the screen, stick out our tongues, vomit and burn down the house. Ever happen to you? Eh? (Oh well it must just be me then...)

Anyway, you know what I mean. The ideas just aren't a flowing. Everything you think of seems to have been done so many times. And, of course, it has.

But here help is at hand. In this occasional series I will be presenting program plans for the popular program forms for you to go away and program. Then you will have a good program which you can say you have written and which, maybe, you could customise to your own requirements. This month I start with that old cowboy, the DATABASE.

People usually groan and then commit suicide when someone says the word database. "Oh my good golly gumdrops," they groan, plunging a carving knife into their chest, "another database." But I promise you that they really aren't as dull as they appear everytime you look at them. With a little thought your database can become an exciting database or even, dare I say, a database.

With a little thought the database can become interesting. Even useful. And heaven itself to use.

THE PLAN

So here follows the plan for the ideal database. When writing your own you may, of course, deviate from this plan to include bits you need or get rid of bits you don't. That's the beauty of the system.

Before you start you must decide upon the user interface which you are going to use. There are numerous which can be used alone or combined.

There is the menu system which is very easy to use for the beginner as all the options currently available are laid out if front of him and all he has to do is press a key.

There is the command line method which, because of its unnecessary complexity, is very little used. The basic idea is that, from a command line, the user enters commands to carry out options which may have been on a menu. For example he may type

```
SEARCH: (3:STEVEN)
```

which may have searched through the file looking at field 3 for the data STEVEN, and so on.

And there is the windows environment. This often incorporates the use of a mouse which, just as often, is not wise as a mouse usually jams up certain keys on the keyboard. And when you are typing data in with the keyboard, it is a bit of a pain having to pull out the mouse and then plug it back in to select an option. But still, windows offer the most attractive method of interfacing. They are clean and uncluttered and very satisfying to use.

Personally I would opt for windows and menus - each time you call a window you are presented with a menu the options from which are selected by pressing a key rather than clicking a mouse.

Of course, some of you may not be able to program windows as it is necessary to save the screen each time one is called. I would not be able to do it were it not for LASER BASIC which is most obliging with regard to saving text screens into memory. So, for those of you who cannot do this, the best choice is probably menus.

It is a good idea, rather than having every option on one menu, to have the options put into special menus which contain other related options. You could have a DISK menu. a PRINTER menu a SEARCH menu and so on. But before you can allocate options to menus you must have options to allocate.

WHAT'S IN?

There are certain options which are absolutely essential. These are listed thus:

CREATE DATABASE : This is where the fields are entered and the number of records required is entered or calculated. Then the database is dimensioned in RAM or is stored as an empty relative file on disk.

ADD RECORDS : This option allows the entry of records which can be stored sequentially or by a calculated mapping function (these will be discussed later). The data, again, is either placed in RAM (in an array) or on disk at the appropriate position.

AMEND RECORDS : Errors always occur so it is vital to have a way of rectifying them. This option should allow for the amendment of any of the fields (which may cause problems with mapped databases).

DELETE RECORDS : If Mr Smith dies or moves to the moon then his subscription must be cancelled. He must be deleted from the database. Again, this may cause problems with mapping functions.

SORT : Perfectionists amongst you will want your database in a nice ordered list. The sort option will do this. Mapped databases cannot accommodate a sort. And with disk based databases, although they can accommodate it, a sort may be impractical.

SEARCH : This is where you can find out how many in your database wear disposable wigs or plastic trusses. The search option is an essential part of any database. They cause no problems with mapped databases (hurrah!!)... well just a minor one (discussed later). They can take ages with disk based bases.

SAVE : You must be able to save your database, otherwise what is the point? Saving is usually automatic with disk based databases. With mapped databases either the entire database must be saved - with all the gaps - or the database must be compressed - which takes some time.

LOAD : What is the use of saving it if you can't load it at a later date? Mapped bases need to be decompressed during loading.

PRINT : Although this option is not really essential, most people like to have it there. It's useful to be able to print out records for, say, mailing lists and what not.

Include all of those options and you will have a database which nobody could really raise their nose at. It has all the elements required for a simple data storage and retrieval system. What more could you want?

Well you could want a great deal more, that's what.

However, before I tell you what more you could have, I will ease your minds about mapping functions.

From the above list of functions you would imagine that mapping functions are simply more bother than they are worth. You can't sort them, you have problems deleting from them and amending them, you need to compress and decompress them etc etc.

But what, I hear you cry, is a mapping function?

MAPPING FUNCTIONS

A mapping function is an equation whose result returns the location into which a piece of data should be put. At the simplest level, this equation: **loc=data**

where loc is the location and data is the entered data. Of course in this instance the value of data would have to be within strict limits and it would have to be numeric. So if you entered HELLO, you would get an error. If you entered 1, then the data, 1, would be stored in location 1 of the database. If you entered 9 then the number 9 would be stored in location 9 and so on.

In practice the equations are rather more complicated, as you might imagine. It is necessary for the equation to equate alphanumeric data rather than just numeric data. So you would have to devise an equation which read, perhaps, the ASCII values of the data or whatever. Then the data is stored at the location determined by the result of the equation.

So with a database you would perform the calculation on a key field - which must be chosen and then is set permanently. This would determine where the data would be put. Then, when searching, the key field only needs to be entered, the same equation is performed so the same result is obtained and the computer knows immediately where to look for the piece of information, rather than having to search through the entire database. The problem is encountered when the user doesn't know what's in the key field. If the key field is not entered then the mapping function will not operate properly so it will become necessary to search through the entire database - just as if it were an ordinary database.

These functions are very nice to use and give the impression of a superfast program, whereas it is the method used that is fast. They are not, however, terribly nice to program. A common occurrence is a clash - where two pieces of different data share the same result when calculated. So they need to share the same location in the array.

The solution to this problem is to set up two segments to the array. One called the mapped table, the other the overflow table. The equation should only return values which fit into the mapped table. So if the mapped table runs from records 1 to 50, the equation should only return values from 1 to 50. then whenever any clashes occur, the extra data is stored in the overflow table and called by pointers. This method of data storage is also called HASH TABLE storage and is expanded upon in a previous magazine article called HASHING IT in the FEB '90 issue of CDU. For C128 owners there was a database program which used mapping functions in the JULY '90 issue of CDU called HASHBASE 128.

The inability to sort and difficulty with amendment and deletion are all drawbacks to the use of mapping functions, but they are incredibly fast for data retrieval and would probably be useful if you need to have immediate access to your data - in a shop for example, if you are asked about your stocks of something you need only enter the product and then immediately have details on screen.

SPACE HAS ONCE AGAIN BEATEN US I'M AFRAID. WE WILL HAVE TO CONTINUE THIS ARTICLE IN THE NEXT ISSUE OF CDU - DUE OUT ON 21st JUNE.

Commodore DISK·USER

Lineage: 58p per word. (+VAT)
Semi display: £11.50 plus VAT per single column centimetre minimum 2cm. Ring for information on series bookings/discounts.

All advertisements in this section must be prepaid.
Advertisements are accepted subject to the terms and conditions printed on the advertisement rate card (available on request).

0908 569819

Send your requirements to:
CDU CLASSIFIED DEPARTMENT, ALPHAVITE PUBLICATIONS LTD., 20 POTTERS LANE, KILN FARM, MILTON KEYNES, MK11 3HF.

SOFTWARE

-THE DEDICATED PD LIBRARY-
IN OUR SECOND YEAR OF SUPPLYING 64 PD, WE STILL CONTINUE TO PROVIDE THE MOST USER FRIENDLY SERVICE AROUND! WE HAVE A WIDE SELECTION OF ALL THE LATEST MIND BLOWING DEMOS THAT REALLY SHOW OFF JUST WHAT THE CBM 64 IS CAPABLE OF DOING. EG. GIANT SPRITE MULTI-PLEXORS, BOBS, VECTORS, PLOTTERS, SINUSES, DYSP'S, DYPP'S & MANY MORE COMPLEX ROUTINES!

MANY DEMOS ARE SO IMPRESSIVE, THEY WOULDN'T EVEN LOOK OUT OF PLACE ON THE AMIGA! EACH ONE IS CREATED BY THE WORLDS BEST DEMO GROUPS & IS PROGRAMMED TO A HIGH STANDARD WITH GREAT GRAPHICS & ORIGINAL MUSIC ALSO INCLUDED. WE ALSO HAVE AN EXCELLENT COLLECTION OF THE BEST PD UTILITIES, MANY OF WHICH ARE BETTER THAN COMMERCIALLY RELEASED ONES! THEY INCLUDE: GRAPHICS, SFX & MUSIC EDITORS, DISK COPIERS, CRUNCHERS, LETTER WRITERS & MANY, MANY MORE. WE EVEN HAVE A FEW PROGRAMMES THAT WILL LET A COMPLETE NOVICE CREATE A DEMO OF THEIR OWN! ALL UTILITIES ARE OF HIGH QUALITY & REALLY HAVE TO BE SEEN TO BELIEVED. IF YOU ENJOY A QUICK GAME OR TWO THEN LOOK NO FURTHER. WE HAVE A RANGE OF THE BEST PD GAMES AROUND, ARCADE, PUZZLE & ADVENTURE. SOME OF WHICH ARE COMMERCIAL QUALITY! WE'RE NOT ONLY THE LEADING SUPPLIER OF 64PD, BUT WE ALSO OFFER FREE EXPERT ASSISTANCE SHOULD ANY PROGRAMMING OR PD PROBLEMS ARISE

FOR A COPY OF OUR LATEST LIST SEND AN SAE TO THE FOLLOWING ADDRESS, ALTERNATIVELY SEND £2.00 FOR A MORE DETAILED COMPUTERISED LIST & A FEW EXTRA DEMOS.
SILVER WING SOFTWARE
185 CALLOWBROOK LANE,
RUBERY,
BIRMINGHAM . B45 9TG

CLASSIFIED COUPON

ALL CLASSIFIED ADVERTISEMENTS MUST BE PRE-PAID.
THERE ARE NO REIMBURSEMENTS FOR CANCELLATIONS.

CDU CLASSIFIED DEPARTMENT, ALPHAVITE PUBLICATIONS,
20 POTTERS LANE, KILN FARM, MILTON KEYNES, MK11 3HF.

RATES:Lineage 58p per word (+VAT). Semi-display: £11.50 (+VAT) per single column cm minimum size 2cm. Series discounts available.

I enclose my Cheque/Postal Order for £................for ....insertions, made payable to Alphavite Publications.
(Delete as necessary)

☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ PLEASE DEBIT MY ACCESS/VISA

CARD NO.:

EXP. DATE:..

£....................................FOR..........................INSERTIONS

Name:...

Address:..

....Post Code:..........................

Daytime Tel No.: ....

Signature: ....Date:.............................

❏ FOR SALE ❏ SOFTWARE ❏ SPECIAL OFFERS ❏ OTHER

ONLY POOLS AND HORSES

Every program written by a mathematician who has spent many years in the betting industry. Programs that utilise the tried and trusted methods of the professional, not pie in the sky theories that fail to pass the test of time.

FOOTBALL BOXFORM Written by a former pools expert for Littlewoods. The program has forecast over 50% more draws than would be expected by chance.
Homes, Aways and draws shown in order of merit and true odds given for every match. Merit tables show at a glance the teams currently in form and those currently having a lean spell. Australian pools program included in the price.

POOLS PLANNER by the same author. Full details given of 369 easily entered block perms ranging from 9 to 73960 lines and from 12 to 56 selections. All are accepted by the pools firms and are checked in seconds by your computer.

RACING BOXFORM Course characteristics (built in to the program) as well as the form of the horses are considered to speedily produce an order of merit for each race. Designed for flexibility allowing users to amend the program if they wish. **Price** still includes the highly acclaimed HANDI CAP WINNER - more than 1000 winners every year - over 25% of them at 5/1 or better. Order two or more and receive **FREE** a program to work out almost any bet. So good its used by bookies,

Prices (Tape) £15.95 each. £25.95 any two. £35.95 all three. For discs please add £2. per program.

Advertised for Six years in the sporting and computing press.
BOXoft CLEVER.. GET THE BEST

BOXOFT (CDU), 65 Allans Meadow,
Neston, South Wirral L64 9SQ
Chque/ P.O./Access/Visa Tel:051-336-2668

VISA

BBC B & MASTER, AMSTRAD CPC & PCW COMMODORE 64/128, SPECTRUMS.

FOR SALE

1520 PRINTER PLOTTER
£45 O.N.O Tel: 0418875054

DEALERS

WORTHING COMPUTERS
Tel: 0903 210861
7 WARWICK ST WORTHING WEST SUSSEX
Stockists of a large range of 8 Bit and 16 Bit Software and Peripherals.

HOME WORKERS WANTED
.(Mailing Envelopes) £3 per Hr.
STAMPED S.A.E. TO
MAT KIZA
27 WOODSIDE PLACE
GLASGOW G37QL.

REPAIRS

REPAIRS
C64, C+4..............................£30. inc.
C128, 1541...........................£40. inc.
Three month warranty, 1 week turnroundd

SPARES

906114.... £9.30	8501.......£11.75
6510.......£11.50	C64 Power Pack £22.00
6526.......£11.95	C64 Cassette unit £22.00
6581.......£14.95	
6569.......£21.50	

OASIS COMPUTER SERVICES
Dept 4C. 14 Ridgeway Road, Salisbury, Wiltshire, SP1 3BU
Tel:(0722) 335061

COMMODORE 64/128 SOFTWARE LIBRARY

☆Life Membership
☆Updates
☆Teenage Mutant Hero Turtle Tape £1.75
☆ 7 Day Hire
☆Tapes From £1.50 to £1.75
Also Disks.

FOR FREE CATALOGUE S.A.E TO -
ACE SOFTWARE LIBRARY,
14 CHIPENDALE COURT, BELFAST, BT10 0DU

...it's dynamite!

POWER CARTRIDGE

FOR YOUR COMMODORE

64/128

"...unbelievable value for money" ZZAPP! Dec 89

SO MUCH FOR SO LITTLE

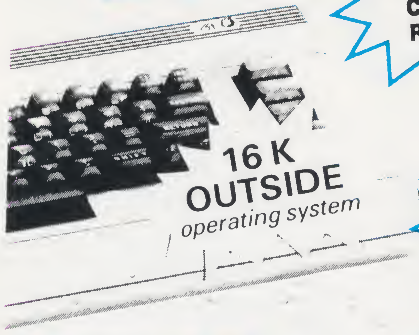
* POWER TOOLKIT
* POWER MONITOR
* TAPE & DISK TURBO
* PRINTERTOOL
* POWER RESET
* TOTAL BACKUP

..."Money well spent" YC/CDU Jan 90

42 Pg Manual - "Damned Good Handbook" CCI Jan 90

AVAILABLE FROM ALL GOOD COMPUTER RETAILERS

TRIED AND TESTED - OVER 100,000 SOLD IN EUROPE



..."highly recommended for C64 users" CCI Jan 90

YOU WILL WONDER HOW YOU EVER MANAGED WITHOUT IT

ONLY £17.30 INC VAT

POWER TOOLKIT

A powerful BASIC-Toolkit (Additional helpful commands) that considerably simplifies programming and debugging.

AUTO	HARDCAT	RENUMBER
AUDIO	HARDCOPY	REPEAT
COLOR	HEX$	SAFE
DEEK	INFO	TRACE
DELETE	KEY	UNNEW
DOKE	PAUSE	QUIT
DUMP	PLIST	MONITOR
FIND	ILOAD	BLOAD

RENUMBER	: *Also modifies all the GOTO's GOSUB's etc. Allows part of a program to be renumbered or displaced.*
PSET	: *Set up of printer type.*
HARDCAT	: *Prints out Directory.*

The toolkit commands can be used in your programs.

DISK TOOL

Using POWER CARTRIDGE you can load up to 6 times faster from disk.
The Disk commands can be used in your own programs.

DLOAD	DVERIFY	DIR
DSAVE	MERGE	DEVICE
DISK		

MERGE	: *Two BASIC programs can be merged into one.*
DISK	: *With DISK you can send commands directly to your disk.*

TAPE TOOL

Using POWER CARTRIDGE you can work up to 10 times faster with your data recorder. The Tape commands can be used in your own programs.

LOAD	SAVE	VERIFY
MERGE	AUDIO	

POWERMON

A powerful machine language monitor that is readily available and leaves all of your Commodore memory available for programming.
Also works in BASIC-ROM, KERNAL and I/O areas.

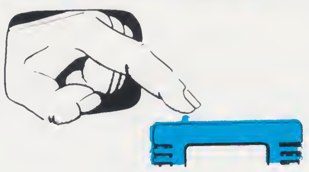
A ASSEMBLE	I INTERPRET	S SAVE
C COMPARE	J JUMP	T TRANSFER
D DIS-ASSEMBLE	L LOAD	V VERIFY
	M MEMORY	W WALK
F FILL	P PRINT	X EXIT
G GO	R REGISTER	$ DIRECTORY
H HUNT		DOS Commands

PRINTERTOOL

The POWER CARTRIDGE contains a very effective Printer-Interface, that self detects if a printer is connected to the Serial Bus or User-Port.
It will print all Commodore characters on Epson and compatible printers.
The printer-interface has a variety of set-up possibilities. It can produce HARDCOPY of screens not only on Serial printers (MPS801, 802, 803 etc) but also on Centronic printers (EPSON, STAR, CITIZEN, PANASONIC, etc).
The HARDCOPY function automatically distingishes between HIRES and LORES. Multi-colour graphics are converted into shades of grey. The PSET functions allow you to decide on Large/Small and Normal/Inverse printing.
The printer PSET functions are:

PSET 0	- *Self detection Serial/Centronics.*
PSET 1	- *EPSON mode only.*
PSET 2	- *SMITH-CORONA mode only.*
PSET 3	- *Turns the printing 90 degrees!!*
PSET 4	- *HARDCOPY setting for MPS802/1526.*
PSET B	- *Bit-image mode.*
PSET C	- *Setting Lower/Upper case and sending Control Codes.*
PSET T	- *All characters are printed in an unmodified state.*
PSET U	- *Runs a Serial printer and leaves the User-port available.*
PSET Sx	- *Sets the Secondary address for HARDCOPY with Serial Bus.*
PSET L1	- *Adds a line-feed, CHR$ (10), after every line.*
PSET L0	- *Switches PSET L1 off*

POWER RESET



On the back of the POWER CARTRIDGE there is a Reset Button. Pressing this button makes a SPECIAL MENU appear on the screen.
This function will work with many programmes.

CONTINUE	- *Allows you to return to your program.*
BASIC	- *Return to BASIC.*
RESET	- *Normal RESET.*
TOTAL BACKUP DISK	- *Saves the contents of the memory onto a Disk. The program can be reloaded later with BLOAD followed by CONTINUE.*
RESET ALL	- *RESET of any program.*
TOTAL BACKUP TAPE	- *As BACKUP DISK but to TAPE.*
HARDCOPY	- *At any moment, prints out a Hardcopy of the screen. Using CONTINUE afterwards you can return to the program.*
MONITOR	- *Takes you into the Machine language Monitor.*

BDL

Bitcon Devices Ltd

88 BEWICK ROAD
GATESHEAD
TYNE AND WEAR
NE8 1RS
ENGLAND

Tel: 091 490 1975 and 490 1919 Fax 091 490 1918
To order: Access/Visa welcome - Cheques or P/O payable to BDL
Price: £17.30 incl. VAT.
UK orders add £1.20 post/pack total - £18.50 incl. VAT.
Europe orders add £2.50. Overseas add £3.50
Scandinavian Mail Order and Trade enquiries to: Bhiab Elektronik, Box 216, Norrtalje 76123, SWEDEN. Tel: + 46 176 18425 Fax: 176 18401
TRADE AND EXPORT ENQUIRIES WELCOME

NEW

THE COMPLETE COLOUR SOLUTION

£179

Vidi ... No 1 in UK & Europe (Leading the way forward)

Amiga Computing: The best Amiga digitiser has had the technicolour treatment. Vidi must be one of the most exciting peripherals you can buy for your Amiga.

Micro Mart: When I first saw Vidi "in the flesh" as it were, at the CES show last September it looked to be the answer to a frustrated Digi View owner's dreams – in fact to see pictures appearing on screen without the customary two minutes wait seemed almost too good to be true. I have consistently produced more good quality pictures in the short time I have had Vidi than I ever did with Digiview.

Zero: Now under normal circumstances cheap usually means poor quality but this is not the case with Rombo. Why? cos Vidi-Amiga is the best digitiser for under £500 and I've tried them all.

Amiga Format: Where qu[illegible] is concerned, Vidi produces so[illegible] the best results I've seen on[illegible] digitiser at any price.

Amiga User International: The latest addition to the Rombokit is called Vidi-RGB and brings this already impressive package to the realms of totally amazing. CONCLUSION: Who will find Vidi-Amiga useful? The answer to this is almost anyone with a video recorder or camera and a passing interest in graphics.

Get the most out of your Amiga by adding:

"The Complete Colour Solution"

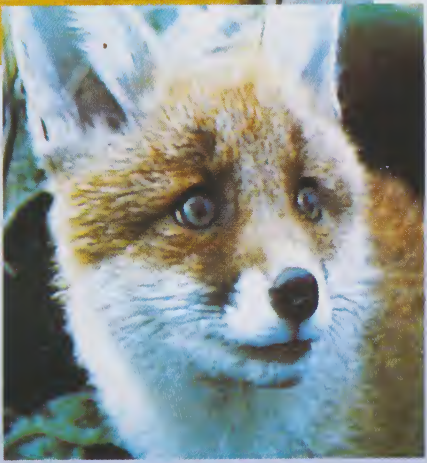
The Worlds ultimate creative leisure product for your Amiga. Capture dynamic high resolution images into your Amiga in less than one second.

And Look No Filters

Images can now be grabbed from either colour video camera, home VCR or in fact any still video source. The traditional method of holding three colour filters in front of your video camera is certainly a thing of the past. Because Vidi splits the RGB colours electronically there are no focussing or movement problems experienced by some of our slower competitors. Lighting is also less of an issue as light is not being shut out by lens filters. Put all this together with an already proven Vidi-Amiga/VidiChrome combination and achieve what is probably the most consistant and accurate high quality 4096 colour images ever seen on the Amiga.

The colour solution is fully compatible with all Amiga's from a standard A500 to the ultimate A3000. No additional RAM is required to get up and running.

You will see from independant review comments that we are undoubtedly their first choice and that was before the complete solution was launched. If you have just purchased your Amiga and are not sure what to buy next, then just read the comments or send for full review and demo disk.



Actual unretouched digitised screenshot

Features ...

- Grab mono images from any video source
- Capture colour images from any still video source.
- Digitise up to 16 mono frames on a 1meg Amiga.
- Animate 16 shade images at different speeds.
- Create windows in both mono & colour.
- Cut & Paste areas from one frame to another.
- Hardware and software brightness & contrast control.
- Choice of capture resolutions standard & Dynamic interlace.
- Full Palette control.
- Add text or draw within art package.

ROMBO Limited

Full colour demonstration disk available for only £1.95 to cover P&P.

6 Fairbairn Road, Livingston, EH54 6TS. Tel: 0506-414631 Fax: 0506-414634